

THÈSE

présentée

A LA FACULTÉ DES SCIENCES ET DES TECHNIQUES DE
L'UNIVERSITÉ DE FRANCHE-COMTÉ

pour obtenir le

DIPLÔME DE DOCTEUR INGÉNIEUR

par

Denis GIEN

Ingénieur ENSMM

**CALCULATEUR PARALLÈLE POUR LA
COMMANDE OPTIMALE EN LIGNE
DE PROCESSUS DYNAMIQUES**

Soutenue le 8 Juillet 1981 devant la Commission d'Examen :

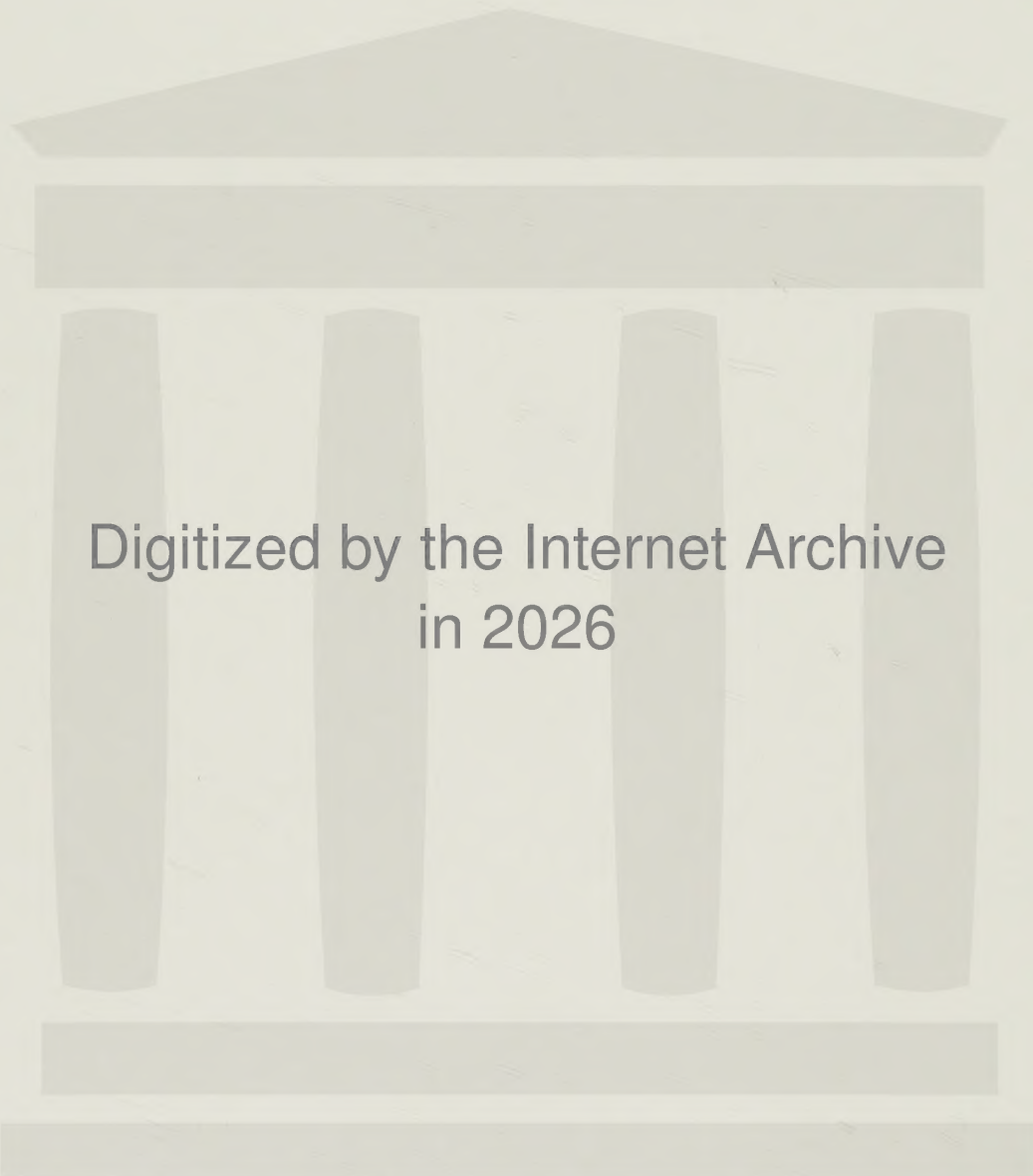
MM. F. LHOTE

Président

P. ANDRÉ
F. BOERI
L. BOLLIET
J.C. MIELLOU
B. NUSSBAUMER
M. TREHEL

} Examineurs

1.° Lise Besançon



Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41546618_0004

THÈSE

présentée

A LA FACULTÉ DES SCIENCES ET DES TECHNIQUES DE
L'UNIVERSITÉ DE FRANCHE-COMTÉ

pour obtenir le

DIPLÔME DE DOCTEUR INGÉNIEUR

PAR

Denis GIEN

INGÉNIEUR ENSMM

CALCULATEUR PARALLELE POUR LA COMMANDE OPTIMALE EN LIGNE DE PROCESSUS DYNAMIQUES

Soutenue le 8 Juillet 1981 devant la commission d'examen :

MM.	F. LHOTE	Président
	P. ANDRE	
	F. BOERI	
	L. BOLLIET	Examineurs
	J.C. MIELLOU	
	B. NUSSBAUMER	
	M. TREHEL	

UNIVERSITÉ DE FRANCHE-COMTÉ BESANÇON

Président : Monsieur le Professeur Jacques ROBERT

FACULTÉ DES SCIENCES ET DES TECHNIQUES

DOYEN

M. le Professeur Jean BULABOIS
Physique Générale-optique

ASSESSEURS

M. le Professeur Bernard MILLET
Botanique
M. Robert GUYETANT, Maître-Assistant
Biol. animale Ecologie

PRÉSIDENTS HONORAIRES

MM. THIEBAUT, LÉVÊQUE

DOYENS HONORAIRES

MM. GLANGEAUD, JACQUEMAIN, CHATELET, THIEBAUT, BERNARD

PROFESSEURS HONORAIRES

MM. BENNETON, CHATELET, MESNAGE, QUANTIN, THEOBALD N., THIRY
TRILLAT, TRONCHET, UEBERSFELD

MM. BANTEGNIE Robert	: Mathématiques
BENILAN Philippe	: Mathématiques
BERNARD Jean	: Chimie Physique
BIDAULT Michel	: Taxonomie expérimentale
BROQUET Paul	: Géologie
BRUCKERT Sylvain	: Ecologie végétale
BULABOIS Jean	: Physique Gle optique
CAPODANNO Pierre	: Mécanique théorique
CERUTTI Ernest	: Chimie appliquée
CHALEAT Raymond	: Mécanique appliquée
CHARDON J-Claude	: Spectroscopie
CHAUVE Pierre	: Géologie minéralogie
COUGNARD Jean	: Mathématiques
DEVIN Claude	: Chimie 1er cycle
DOBREMEZ Jean-Fr.	: Biologie
DUBOUCHET Jacques	: Physiologie végétale
GALATRY Louis	: Physique moléculaire
GAUDEMER Yves	: Biochimie
GIRARDET Claude	: Physique Moléculaire
GOMOT Lucien	: Zoologie-Embryologie
GOUARNE René	: Mathématiques
GRAS Georges	: Mathématiques
GREMILLARD Jean	: Mathématiques
KARCHE J-Paul	: Pétrographie Minéralogie
LALLEMENT Gérard	: Mécanique appliquée
LAUDE Bernard	: Chimie organique
LESAINTE Pierre	: Mathématiques I

LHOTE François	: Automatique
MERIGOUX Henri	: Cristallographie et Synthèses minérales
MIELLOU J-Claude	: Analyse numérique
MILLET Bernard	: Botanique
MONTAGNER Hubert	: Psychophysiologie
MOREELS Guy	: Astronomie
OLIVIER Marchel	: Electroniq. quantique
OYTANA Claude	: Mécanique appliquée
PAGETTI Jacques	: Corrosion et traitements de surface
PARIZET Jean	: Mathématiques
PLUVINAGE Philippe	: Physique théorique
POTIER Robert	: Physique des solides
RANGHEARD Yves	: Géologie
RIPPLINGER Jean	: Physiologie animale
ROBERT Daniel	: Physique
ROBERT Guy	: Electrochimie
ROBERT Jacques	: Mathématiques I
THEOBALD J-Gérard	: Spectroscopie hertzienne
THIEBAUT Jean	: Pétrogr. minéralogie
TREHEL Michel	: Informatique
VERNEAUX Jean	: Zoologie-Embryologie
VIENOT Jean-Charles	: Physique Gle optique
WEIL Michel	: Mathématiques

Responsable des services administratifs :
M. Jean-Marie RUNGE

Professeurs des Instituts Universitaires de Technologie dont les activités de Recherche sont rattachées à la Faculté des Sciences et des Techniques de Besançon

I.U.T. de BESANÇON

MM. GENTYL Claude	Spécialité : Physique
MARTINET André	Spécialité : Chimie
SONZOGNO Robert	Spécialité : Mécanique

E.N.S.M.M.

MM. AVRAMESCO	Spécialité : Mathématiques
BESSION	Spécialité : Electronique
ROQUES-CARMES	Spécialité : Chimie minérale

I.U.T. de BELFORT

MM. KAUFFMANN J-Marie	Spécialité : Electrotechnique
MANDRET Gérard	Spécialité : Electronique
OIKNINE Claude	Spécialité : Mécanique
PRENEL Jean-Pierre	Spécialité : Physique
STAMON Georges	Spécialité : Informatique

AVANT - PROPOS

Les travaux présentés dans ce mémoire ont été réalisés au Laboratoire d'Automatique de l'Ecole Nationale Supérieure de Mécanique et des Microtechniques de Besançon, et n'auraient pu être menés à bien sans l'aide et le soutien d'un grand nombre de personnes.

Je voudrais tout d'abord remercier les membres du jury :

- Monsieur LHOTE, Directeur du Laboratoire d'Automatique de l'Ecole Nationale Supérieure de Mécanique et des Microtechniques, pour m'avoir accueilli dans son laboratoire et pour avoir suivi l'ensemble de ce travail ;
- Monsieur ANDRE, Maître-assistant à l'Ecole Nationale Supérieure de Mécanique et des Microtechniques, pour avoir proposé l'étude d'un calculateur spécialisé pour la commande optimale et pour les conseils qu'il m'a donné pour la réalisation du calculateur ;
- Monsieur BOERI, Professeur à l'université de Nice, pour avoir bien voulu accepter d'examiner ce travail ;
- Monsieur BOLLIET, Professeur à l'Université de Grenoble, pour l'honneur qu'il me fait en participant à ce jury ;
- Monsieur MIELLOU, Professeur à l'Université de Franche-Comté, pour l'intérêt qu'il a montré pour ce travail ;
- Monsieur NUSSBAUMER, Ingénieur à la Société C.I.I. H.B. , pour avoir accepté de participer à ce jury ;

- Monsieur TREHEL, Professeur à l'Université de Franche-Comté,
pour son aimable participation à ce jury.

Je remercie également les personnes qui ont participé aux
réalisations matérielles :

- Monsieur FOLIN, pour le câblage du calculateur ;

- Madame CÔTE ainsi que Messieurs RENAUDOT et DEMOULIN pour
le soin qu'ils ont apporté à la réalisation du mémoire.

S O M M A I R E

I - COMMANDE OPTIMALE

1 - DEFINITION DU PROBLEME

1.1. Le modèle

1.1.1. Modèle continu

1.1.2. Modèle récurrent

1.2. Le critère

1.2.1. Critère continu

1.2.2. Critère récurrent

1.3. Problème à résoudre

1.3.1. Domaine d'admissibilité

1.3.2. Horizon

2 - RESOLUTION

2.1. Programmation mathématique

2.2. Equations d'optimalité

2.3. Méthode de résolution

3 - MODELE LINEAIRE STATIONNAIRE, CRITERE QUADRATIQUE ET CONTRAINTES

3.1. Modèle et objectif

3.2. Conditions nécessaires et suffisantes d'optimalité

3.3. Equations d'optimalité

4 - CALCUL DE LA LOI DE COMMANDE OPTIMALE

4.1. Technique de calcul

4.2. Convergence

4.2.1. Rappel

4.2.2. Convergence du calcul de la loi de commande optimale

4.3. Paramètre de relaxation

5 - CONDUITE DU PROCESSUS

5.1. Conduite en boucle ouverte

5.2. Conduite en boucle fermée

CONCLUSION

II - DECOMPOSITION DES CALCULS ET COMMANDE DECENTRALISEE AVEC COORDINATION

1 - SYSTEMES INTERCONNECTES

1.1. Le modèle

1.2. Le critère

1.3. Exemples de systèmes interconnectés

2 - DECOMPOSITION DES CALCULS

2.1. Recherche du point fixe d'une application par un
algorithme de relaxation chaotique à retards

2.1.1. Itérations chaotiques

2.1.2. Itérations chaotiques à retards

2.1.3. Convergence

2.2. Application au calcul de la loi de commande optimale

2.2.1. Décomposition de l'application F

2.2.2. Décomposition des équations d'optimalité

2.2.3. Problèmes locaux et coordination

2.2.4. Calcul hiérarchisé et décomposition horizontale

3 - COMMANDE DECENTRALISEE

3.1. Méthode optimale

3.2. Méthodes sous-optimales

3.2.1. Coordination statique

3.2.2. Coordination dynamique

3.2.2.1. Fonctionnement synchrone

3.2.2.2. Fonctionnement asynchrone

3.3. Méthode mixte

CONCLUSION

III - SYSTEMES INFORMATIQUES POUR LA COMMANDE OPTIMALE

1 - RAPPELS

1.1. Calcul parallèle

1.1.1. Les calculateurs multiprocesseurs

1.1.2. Les calculateurs scientifiques matriciels

1.2. Systèmes distribués

2 - CALCULATEUR PARALLELE POUR LE CALCUL DE LA LOI DE COMMANDE OPTIMALE

2.1. Calculateur multiprocesseur

2.2. Calculateur matriciel

2.3. Calculateur vecteur spécialisé

2.4. Système multiprocesseur vecteur

3 - SYSTEME DECENTRALISE DE COMMANDE OPTIMALE

3.1. Architecture du système

3.2. Calcul de la loi de commande

IV - CARACTERISTIQUES DU CALCULATEUR VECTEUR

1 - PROBLEME A TRAITER

2 - CALCULATEUR VECTEUR

3 - DIMENSION DU CALCULATEUR

4 - PRECISION

4.1. Représentation des nombres

4.2. Cadrage des variables

4.3. Erreur sur la solution

4.4. Estimation de l'erreur commise au cours d'une itération

5 - EXPERIENCES PRELIMINAIRES

V - ARCHITECTURE DU CALCULATEUR VECTEUR

1 - GENERALITES

1.1. Calculateur microprogrammé

1.2. Calculateur vecteur

2 - L'UNITE DE COMMANDE

2.1. La mémoire et le registre pipe-line

2.2. Le séquenceur

2.3. Commande de l'adressage et des échanges

2.4. Commande des autres fonctions

3 - UNITE DE TRAITEMENT

3.1. L'unité de calcul

3.2. Rangement et transfert des données

4 - MISE EN OEUVRE

VI - PROGRAMMATION

1 - CALCUL DE LA COMMANDE OPTIMALE

2 - MISE EN OEUVRE SUR LE CALCULATEUR VECTEUR

2.1. Calcul vectoriel

2.2. Multiplication et précision

2.3. Test de convergence et contraintes

3 - EXPLOITATION

3.1. La programmation

3.2. Exploitation expérimentale du calculateur

3.3. Mise au point du calculateur

VII - PERFORMANCES

1 - REALISATION DU PROTOTYPE

1.1. Choix des composants

1.2. Le câblage

2 - PERFORMANCES

2.1. Evaluation de la vitesse de calcul

2.1.1. Nombre d'itérations

2.1.2. Nombre d'instructions par itération

2.1.3. Durée du cycle

2.2. Performances réelles du calculateur actuellement réalisé

2.2.1. Vitesse

2.2.2. Précision

2.3. Performances du calculateur complet

2.4. Amélioration des performances

3 - CONDUITE D'UNE SIMULATION DE PROCESSUS

3.1. Conduite centralisée

3.1.1. Commande d'un processus parfait

3.1.2. Régulation d'un processus perturbé

3.1.3. Commande d'un processus instable

3.2. Simulation de conduite décentralisée

CONCLUSION

NOTATIONS

x^n : vecteur d'état à l'instant n

λ^n : vecteur d'état adjoint à l'instant n

u^n : vecteur de commande entre les instants n et $n+1$

y^n : vecteur de sortie à l'instant n

$\{x^n\}$: trajectoire d'état

$\{\lambda^n\}$: trajectoire adjointe

$\{u^n\}$: loi de commande

$\{y^n\}$: évolution de la sortie

x : valeur de x au cours d'un calcul

$\hat{x}$: valeur optimale de x , résultat du calcul

$\bar{x}$: valeur de x au cours d'un calcul approché

$\tilde{x}$: valeur de x après troncature ou arrondi

$\underline{x}$: valeur de l'état réel du processus

$\tilde{\underline{x}}$: prédiction de la valeur $\underline{x}$

A, B, C : modèle linéaire d'un processus

P, Q : matrices de pondération d'un critère quadratique

$J(x,u,n)$: critère

$r(x,u,n)$: coût instantané

z : variable réelle vectorielle

$f(z)$: fonction de z

$e(x,u,n)$: modèle, cas général

$g(z)$: contrainte égalité

$h(z)$: contrainte inégalité

$s(x,u,n)$: réponse, cas général

$\phi(z,\lambda,\mu)$: fonction augmentée, optimisation de $f(z)$

λ : paramètre de Lagrange

μ : paramètre de Kühn et Tücker

ω : paramètre de relaxation

d : paramètre de contraction

D : paramètre de contraction vectoriel

ϵ : erreur de calcul

δ : test de convergence

$F(U)$: application dont le point fixe est solution du problème

$F'(W)$: cas du calcul décentralisé

$F_\omega(U)$, $F'_\omega(U)$: avec paramètre de relaxation

e_i^n : terme de couplage état

f_i^n : terme de couplage adjoint

P_i : sous processus

C_i : calculateur, processeur local

$|z|$: valeur absolue de z (scalaire)

$||z||$: norme (euclidienne) du vecteur z

$|||z|||$: norme vectorielle du vecteur z

M^T : transposée de M

$\rho(M)$: rayon spectral de M

$||M||$: norme de M , $\rho(M^T M)$

I N T R O D U C T I O N

Les techniques de régulation classiques sont mal adaptées à la commande d'installations industrielles complexes. La commande optimale en boucle ouverte n'est satisfaisante que pour des processus peu perturbés dont le modèle est connu avec précision. La commande optimale en boucle fermée n'est généralement possible que pour un système linéaire non contraint et un critère quadratique.

Les études menées précédemment au Laboratoire d'Automatique de Besançon [2] ont conduit à la commande optimale en "boucle fermée par intermittence" de procédés par ordinateur hybride [6]. Malheureusement le coût et le manque de fiabilité d'éléments analogiques performants ne permettent pas d'utiliser les résultats obtenus par une application réelle.

Nous nous sommes donc dirigés vers la réalisation d'un ordinateur numérique parallèle destiné à la commande de processus dynamiques rapides. En vue d'établir la faisabilité de ce mode de commande nous nous sommes limités, dans un premier temps, au problème linéaire avec contraintes sur la commande et critère quadratique, le principe de calcul restant valable pour la résolution de problèmes non-linéaires.

Les différents éléments d'une installation industrielle étant répartis sur le site nous envisageons l'utilisation de ce ordinateur dans un système informatique distribué adapté à la conduite de processus interconnectés.

Dans le premier chapitre, après un rappel des conditions d'optimalité, nous exposons une méthode de calcul de la commande simple à mettre en oeuvre. Nous explicitons ensuite l'algorithme proposé dans le cas linéaire avec contraintes sur la commande et critère quadratique. Nous envisageons enfin la conduite d'un processus en boucle fermée par intermittence.

Le deuxième chapitre est consacré aux processus interconnectés. D'une part nous étudions la décomposition du calcul de la loi de commande, d'autre part nous proposons une méthode de décentralisation du système de commande avec coordination horizontale des actions locales.

Le troisième chapitre traite des architectures informatiques permettant le calcul de la loi de commande optimale en un temps très bref ainsi que des systèmes informatiques distribués destinés à la conduite de processus interconnectés répartis sur le site industriel.

Nous abordons ensuite la description du calculateur vecteur spécialisé pour le calcul de la commande. Nous donnons les caractéristiques du calculateur dans le quatrième chapitre puis nous exposons l'architecture et la programmation dans les cinquième et sixième chapitres.

Le dernier chapitre est consacré à l'étude des performances du calculateur réalisé ainsi qu'aux premières expériences destinées à évaluer les possibilités de conduite de processus par le système proposé.

I - COMMANDE OPTIMALE

1 - DEFINITION DU PROBLEME

1.1. Le modèle

1.1.1. Modèle continu

1.1.2. Modèle récurrent

1.2. Le critère

1.2.1. Critère continu

1.2.2. Critère récurrent

1.3. Problème à résoudre

1.3.1. Domaine d'admissibilité

1.3.2. Horizon

2 - RESOLUTION

2.1. Programmation mathématique

2.2. Equations d'optimalité

2.3. Méthode de résolution

3 - MODELE LINEAIRE STATIONNAIRE, CRITERE QUADRATIQUE ET CONTRAINTES

3.1. Modèle et objectif

3.2. Conditions nécessaires et suffisante d'optimalité

3.3. Equations d'optimalité

4 - CALCUL DE LA LOI DE COMMANDE OPTIMALE

4.1. Technique de calcul

4.2. Convergence

4.2.1. Rappel

4.2.2. Convergence du calcul de la loi de commande optimale

4.3. Paramètre de relaxation

5 - CONDUITE DU PROCESSUS

5.1. Conduite en boucle ouverte

5.2. Conduite en boucle fermée

CONCLUSION

La conduite, par des méthodes classiques, de processus industriels de plus en plus complexes, répond mal aux impératifs économiques actuels. Il ne suffit plus de contrôler le système mais il est nécessaire de trouver une procédure permettant de le faire de la "meilleure" façon possible. C'est l'objet de la commande optimale.

La démarche à suivre comporte trois phases: la formulation mathématique du problème, la recherche d'une méthode de résolution et la mise en oeuvre d'un principe de conduite du processus.

1 - DEFINITION DU PROBLEME

Le problème mathématique de commande optimale est formulé par la détermination d'un modèle du processus et par le choix d'un objectif sous la forme d'un "coût" à minimiser.

1.1. Le modèle

Le processus est vu de l'extérieur par ses entrées (ou commandes) ainsi que par ses sorties. Les sorties dépendent des entrées mais aussi de l'évolution passée du processus. Le processus sera représenté par trois vecteurs à composantes réelles:

- le vecteur d'entrée u : $u \in \mathbb{R}^J$
- le vecteur de sortie y : $y \in \mathbb{R}^K$
- le vecteur d'état interne x : $x \in \mathbb{R}^I$

résumant le passé du processus.

Le modèle va permettre, connaissant l'état x à un instant donné de déterminer l'évolution future de la sortie y en fonction de celle de l'entrée u .

1.1.1. Modèle continu

Le modèle mathématique représentant le plus exactement le processus est souvent un modèle continu.

Pour la variable t appartenant à l'horizon $[0, T] \subset \mathbb{R}$ notons :

$u(t)$ loi de commande
 $y(t)$ évolution de la sortie
 $x(t)$ trajectoire d'état

Le modèle du processus est défini par l'équation différentielle

$$(1.1) \quad \dot{x}(t) = e(x(t), u(t), t) \quad x(0) = x_0$$

et par l'équation algébrique :

$$(1.2) \quad y(t) = s(x(t), u(t), t)$$

1.1.2. Modèle récurrent

La conduite du processus par un calculateur numérique consiste, à des instants discrets, à observer la sortie puis à modifier la commande. Vu du calculateur le processus est naturellement représenté par un modèle récurrent.

L'indice temporel n appartenant à l'horizon $[0, N] \subset \mathbb{N}$ notons :

$\{u^n\}$ loi de commande
 $\{y^n\}$ évolution de la sortie
 $\{x^n\}$ trajectoire d'état

Le modèle du processus est maintenant défini par l'équation de récurrence :

$$(1.3) \quad x^{n+1} = e(x^n, u^n, n) \quad n = 0, \dots, N-1 \quad x^0 = x_0$$

et par l'équation :

$$(1.4) \quad y^n = s(x^n, u^{n-1}, n) \quad n = 1, \dots, N$$

Remarque 1 : x^n désigne l'état à l'instant n , et u^n la commande dans l'intervalle suivant l'instant n .

Remarque 2 : La détermination d'un modèle récurrent associé à un processus peut être effectuée par identification directe, ou par discrétisation d'un modèle continu (supposé connu) du processus.

1.2. Le critère

L'objectif est de minimiser une fonction "coût" (ou critère) combinant, par exemple, d'une part l'écart de la sortie avec une consigne et d'autre part la consommation d'énergie.

1.2.1. Critère continu

Le critère associé à un modèle continu est une fonctionnelle de la forme :

$$(1.5) \quad J = \int_0^T r(x(t), u(t), t) dt + s(x(T), T)$$

1.2.2. Critère récurrent

Pour un modèle récurrent le critère est une fonction :

$$(1.6) \quad J = \sum_{n=1}^N r(x^n, u^{n-1}, n)$$

Le coût instantané $r(x^n, u^{n-1}, n)$ dépend de x^n , étant à l'instant n , et de u^{n-1} , commande appliquée dans l'intervalle de temps précédent l'instant n .

1.3. Problème à résoudre

Le problème consiste à rechercher une loi de commande "admissible" minimisant le critère pour un horizon donné, l'état final n'étant pas imposé a priori.

1.3.1. Domaine d'admissibilité

Nous nous limiterons au cas de contraintes inégalités :

$$(1.7) \quad h_1(u) \leq 0, \dots, h_L(u) \leq 0 \quad L \leq J$$

Le domaine d'admissibilité est alors défini comme :

$$U_{ad} = \{u / h_i(u) \leq 0, \quad i = 1, \dots, L\}$$

1.3.2. Horizon

Nous nous intéresserons à un horizon du type glissant c'est-à-dire que nous calculerons à un instant t la loi de commande optimale pour l'horizon $[t, t + T]$. Il est facile (notamment pour un processus à modèle invariant) de se ramener par changement de variable à l'horizon $[0, T]$ ou $[0, N]$ pour un modèle récurrent.

Nous utiliserons maintenant la formulation récurrente du problème, plus simple et s'adaptant mieux à l'utilisation de techniques numériques.

Le problème consiste alors à rechercher le minimum du critère (1.6), fonction des variables $x^1, \dots, x^N, u^0, \dots, u^{N-1}$, sous les contraintes égalités (1.3) et les contraintes inégalités (1.7). Nous sommes en présence d'un problème de "programmation mathématique".

2 - RESOLUTION

Le problème étant posé nous allons rechercher les équations caractérisant la solution.

2.1. Programmation mathématique [7]

On recherche le minimum d'une fonction $f(z)$ de $\mathbb{R}$ dans $\mathbb{R}$

sous les contraintes égalités $g_i(z) = 0$ $i = 1, \dots, p$ $p \leq m$
 (que nous noterons $g(z) = 0$) et les contraintes inégalités $h_i(z) \leq 0$
 $i = 1, \dots, q$ $q \leq m$.

Supposons que f soit continuellement différentiable et que g et h admettent des dérivées partielles continues, G_z de rang p et H_z de rang q (contraintes indépendantes).

Théorème [10] : Si, sous les hypothèses précédentes, la fonction $f(z)$ admet au point $\hat{z}$ un minimum relatif dans le domaine défini par les contraintes $g_i(z) = 0$ et $h_i(z) \leq 0$ alors il existe des paramètres de Lagrange λ_i et des paramètres de Kühn et Tücker $\mu_i \geq 0$ tels que la fonction augmentée

$$(2.1) \quad \phi(z, \lambda, \mu) = f(z) + \sum_{i=1}^p \lambda_i g_i(z) + \sum_{i=1}^q \mu_i h_i(z)$$

vérifie :

$$(2.2) \quad \begin{cases} \phi_z(\hat{z}, \lambda, \mu) = 0 \\ \mu_i h_i(\hat{z}) = 0 \end{cases}$$

Si de plus, f est strictement convexe et si le domaine défini par les contraintes est convexe (les fonctions g_i sont linéaires et les fonctions h_i sont convexes) les conditions précédentes sont des conditions nécessaires et suffisantes pour que $\hat{z}$ soit le minimum absolu de f . Les équations caractérisant la solution s'écrivent alors :

$$\begin{cases} g(z) = 0 \\ \phi_z(z) = 0 \\ \mu_i h_i(z) = 0, \quad \mu_i \geq 0, \quad h_i(z) \leq 0 \end{cases}$$

2.2. Equations d'optimalité

Nous allons appliquer ces résultats au problème de commande optimale.

Nous cherchons à minimiser le critère (1.6) :

$$J(x^1, \dots, x^N, u^0, \dots, u^{N-1}) = \sum_{n=1}^N r(x^n, u^{n-1}, n)$$

sous les contraintes égalités (1.3) :

$$x^{n+1} = e(x^n, u^n, n) \quad n = 0, \dots, N-1$$

et les contraintes inégalités (sous la forme générale) (1.7) :

$$h(x^n, u^{n-1}, n) \leq 0 \quad n = 1, \dots, N$$

Ecrivons la fonction augmentée ϕ (2.1) :

$$\begin{aligned} \phi(x^1, \dots, x^N, u^0, \dots, u^{N-1}) &= \sum_{n=1}^N r(x^n, u^{n-1}, n) + \\ &\sum_{n=0}^{N-1} \lambda^n T (e(x^n, u^n, n) - x^{n+1}) + \sum_{n=1}^N \mu^n T h(x^n, u^{n-1}, n) \end{aligned}$$

ou encore :

$$\begin{aligned} (2.3) \quad \phi(x^1, \dots, x^N, u^0, \dots, u^{N-1}) &= \sum_{n=1}^N \{ r(x^n, u^{n-1}, n) + \\ &\lambda^{n-1} T (e(x^{n-1}, u^{n-1}, n-1) - x^n) + \mu^n T h(x^n, u^{n-1}, n) \} \end{aligned}$$

Sous les hypothèses énoncées au début du paragraphe 2.1 les conditions nécessaires pour que J admette un minimum au point $(\hat{x}^1, \dots, \hat{x}^N, \hat{u}^0, \dots, \hat{u}^{N-1})$ s'écrivent :

$$\phi_x = 0 \quad \left\{ \begin{array}{l} \lambda^{n-1} = r_x(\hat{x}^n, \hat{u}^{n-1}, n) + E_x^T \lambda^n + H_x^T \mu^n \quad n = 1, \dots, N-1 \\ \lambda^{N-1} = r_x(\hat{x}^N, \hat{u}^{N-1}, N) + H_x^T \mu^N \end{array} \right.$$

$$\phi_u = 0 \quad \left\{ \begin{array}{l} r_u(\hat{x}^n, \hat{u}^{n-1}, n) + E_u^T \lambda^{n-1} + H_u^T \mu^n = 0 \quad n = 1, \dots, N \\ \mu_i^n h_i(\hat{x}^n, \hat{u}^{n-1}, n) = 0 \quad \mu_i^n \geq 0 \quad h_i^n \leq 0 \\ n = 1, \dots, N \quad i = 1, \dots, L \end{array} \right.$$

Les lois de commande optimales $(\hat{u}^0, \dots, \hat{u}^{N-1})$ et accessoirement les trajectoires d'état $(\hat{x}^1, \dots, \hat{x}^N)$ correspondantes sont parmi les solutions d'un système comprenant :

- une équation de récurrence directe à condition initiale:

$$x^{n+1} = e(x^n, u^n, n) \quad n = 0, \dots, N-1 \quad x^0 = x_0$$

- une équation de récurrence inverse à condition finale:

$$\lambda^{n-1} = r_x(x^n, u^{n-1}, n) + E_x^T \lambda^n + H_x^T \mu^n \quad n = N, \dots, 1 \quad \lambda^N = 0$$

- des équations:

$$\left\{ \begin{array}{l} 0 = r_u(x^n, u^{n-1}, n) + E_u^T \lambda^{n-1} + H_u^T \mu^n \quad n = 1, \dots, N \\ 0 = \mu_i^n h_i(x^n, u^{n-1}, n) \quad n = 1, \dots, N \quad i = 1, \dots, L \\ h_i^n \leq 0 \quad \mu_i^n \geq 0 \end{array} \right.$$

2.3. Méthode de résolution [15]

Nous nous proposons d'obtenir une des solutions de ce système d'équations en utilisant un algorithme de relaxation.

A partir d'une estimation initiale $\{u^n\}^0$ de la loi de commande on obtient la trajectoire d'état correspondante $\{x^n\}^0$ par l'équation de récurrence (2.4). En reportant dans l'équation de récurrence inverse (2.5) les trajectoires $\{u^n\}^0$ et $\{x^n\}^0$ et si nécessaire (contraintes sur l'état) une estimation initiale $\{\mu^n\}^0$ on calcule la trajectoire adjointe $\{\lambda^n\}^0$. Enfin on résout les équations (2.6) pour obtenir une nouvelle loi de commande $\{u^n\}^1$ ainsi que la trajectoire $\{x^n\}^1$. Le calcul est alors réitéré jusqu'à la convergence vers une solution du système.

Pour chaque problème à résoudre il est nécessaire de déterminer comment choisir $\{u^n\}$ pour que le résultat soit un minimum absolu et de vérifier la convergence de l'algorithme.

Nous allons étudier de façon plus précise cet algorithme pour la résolution d'un problème particulier.

3 - MODELE LINEAIRE STATIONNAIRE, CRITERE QUADRATIQUE ET CONTRAINTES

Le modèle linéaire représente souvent imparfaitement la réalité physique, mais nous le considérerons en tant que linéarisation d'un modèle plus complexe au voisinage d'un point de régulation. L'intérêt de ce modèle réside dans sa simplicité permettant un calcul rapide de la loi de commande optimale qui, comme nous le verrons, est ici unique.

3.1. Modèle et objectif

$$(3.1) \quad \begin{cases} X^{n+1} = A X^n + B U^n & n = 0, \dots, N-1 \\ Y^n = C X^n & n = 1, \dots, N \end{cases} \quad X^0 = X_0$$

avec :

$$X \in \mathbb{R}^I, \quad U \in \mathbb{R}^J, \quad Y \in \mathbb{R}^K \quad \begin{matrix} J \leq I \\ K \leq I \end{matrix}$$

A, B, C étant des matrices à coefficients réels de dimensions respectives I.I, I.J, K.I.

Au problème de la régulation du système au voisinage du point de consigne Y_d nous associons un critère pénalisant l'écart entre la sortie Y et Y_d ainsi que les variations de la commande U autour de la valeur U_d permettant de maintenir, en régime permanent, le processus au point de fonctionnement. En notant X_d l'état correspondant nous obtenons :

$$X_d = A X_d + B U_d$$

$$Y_d = C X_d$$

Prenons un critère quadratique :

$$(3.2) \quad J = \frac{1}{2} \sum_{n=1}^N \left(Y^n - Y_d \right)^T R \left(Y^n - Y_d \right) + \left(U^{n-1} - U_d \right)^T Q \left(U^{n-1} - U_d \right)$$

Posons :

$$x = X - X_d, \quad u = U - U_d, \quad y = Y - Y_d$$

$$P = C^T R C$$

Le problème est alors défini par l'équation d'état :

$$(3.3) \quad x^{n+1} = A x^n + B u^n \quad n = 0, \dots, N-1 \quad x^0 = x_0$$

et par le critère :

$$(3.4) \quad J = \frac{1}{2} \sum_{n=1}^N \left(x^n \right)^T P x^n + \left(u^{n-1} \right)^T Q u^{n-1}$$

Supposons en outre que :

P est une matrice à coefficients réels définie positive

Q est une matrice à coefficients réels définie non négative

(P est de dimension $I.I$, Q de dimension $J.J$).

Nous nous limiterons à des contraintes de la forme :

$$u_{j \min} \leq u_j \leq u_{j \max} \quad j = 1, \dots, J$$

$$(u_{j \min} < u_{j \max})$$

Ces contraintes délimitent un polyèdre convexe :

$$U_{ad} = \left\{ u / h_j(u) \leq 0 \quad j = 1, \dots, J \right\}$$

avec :

$$h_j(u) = (u_j - u_{j \max}) (u_j - u_{j \min})$$

3.2. Conditions nécessaires et suffisantes d'optimalité

Toutes les conditions de différentiabilité et de convexité étant réunies la loi de commande optimale $\{\hat{u}^n\}$ est la solution du système :

$$(3.6) \quad x^{n+1} + A x^n + B u^n \quad n = 0, \dots, N-1 \quad x^0 = x_0 \text{ équation d'état}$$

$$(3.7) \quad \lambda^{n-1} = P x^n + A^T \lambda^n \quad n = N, \dots, 1 \quad \lambda^N = 0 \text{ équation adjointe}$$

$$(3.8) \quad \left\{ \begin{array}{l} \alpha \left\{ \begin{array}{l} 0 = Q u^n + B^T \lambda^n + H_u^T \mu^n \quad n = 0, \dots, N-1 \\ \text{avec :} \\ H_u^T = \begin{bmatrix} 2 u_1 - u_{1 \min} - u_{1 \max} & & 0 \\ & \ddots & \\ 0 & & 2 u_J - u_{J \min} - u_{J \max} \end{bmatrix} \end{array} \right. \\ \beta \left\{ \begin{array}{l} 0 = \mu_i^n (u_i^n - u_{i \min}) (u_i^n - u_{i \max}) \\ \text{avec :} \\ \mu_i^n \geq 0 \quad u_{i \min} \leq u_i^n \leq u_{i \max} \end{array} \right. \end{array} \right. \quad \begin{array}{l} n = 0, \dots, N-1 \\ i = 1, \dots, J \end{array}$$

Si, de plus, nous supposons que la matrice Q est diagonale les équations (3.8) peuvent se reformuler beaucoup plus simplement (annexe I.1) :

$$(3.9) \quad \left\{ \begin{array}{l} \text{posons : } v^n = - Q^{-1} B^T \lambda^n \quad (Q \text{ définie positive}) \\ \text{alors} \quad \begin{array}{ll} \text{si } u_{i \min} \leq v_i^n \leq u_{i \max} & u_i^n = v_i^n \\ \text{si } v_i^n > u_{i \max} & u_i^n = u_{i \max} \\ \text{si } v_i^n < u_{i \min} & u_i^n = u_{i \min} \end{array} \end{array} \right.$$

3.3. Equations d'optimalité

Le système d'équations caractérisant la solution $\hat{u}^n$ peut prendre deux formes pratiques. La première s'écrit :

Une équation de récurrence directe :

$$x^{n+1} = A x^n + B u^n \quad n = 0, \dots, N-1 \quad x^0 = x_0$$

Une équation de récurrence inverse :

$$\lambda^{n-1} = P x^n + A^T \lambda^n \quad n = N, \dots, 1 \quad \lambda^N = 0$$

(3.10) Une expression linéaire donnant v^n

$$\text{soit : } v^n = -Q^{-1} B^T \lambda^n \quad n = 0, \dots, N-1$$

Une relation non linéaire donnant u^n

$$\left\{ \begin{array}{ll} \text{si } u_i \min \leq v_i^n \leq v_i \max & u_i^n = v_i^n \\ \text{si } v_i^n > v_i \max & u_i^n = u_i \max \\ \text{si } v_i^n < v_i \min & u_i^n = u_i \min \end{array} \right.$$

Nous en proposons une seconde, que l'on peut obtenir en éliminant les variables x et λ .

$$x^n = A^n x_0 + \sum_{\ell=0}^{n-1} A^{n-1-\ell} B u^\ell \quad n = 0, \dots, N$$

$$\lambda^n = \sum_{\ell=n+1}^{N-1} A^{T(\ell-(n+1))} P x^\ell \quad n = 0, \dots, N-1$$

$$\lambda^n = \sum_{\ell=n+1}^N A^{T(\ell-(n+1))} P \left(A^\ell x_0 + \sum_{k=0}^{\ell-1} A^{\ell-1-k} B u^k \right) \quad n = 0, \dots, N$$

$$v^n = -Q^{-1} B^T \sum_{\ell=n+1}^N A^{T(\ell-(n+1))} P \sum_{k=0}^{\ell-1} A^{\ell-1-k} B u^k$$

$$-Q^{-1} B^T \sum_{\ell=n+1}^N A^{T(\ell-(n+1))} P A^\ell x_0 \quad n = 0, \dots, N-1$$

Pour écrire cette dernière sous forme plus condensée notons :

$$U = \begin{bmatrix} u^0 \\ \vdots \\ u^{N-1} \end{bmatrix} \quad V = \begin{bmatrix} v^0 \\ \vdots \\ v^{N-1} \end{bmatrix} \quad \text{vecteur de } \mathbb{R}^{JN}$$

Nous pouvons alors écrire :

$$V = MU + N x_0 = G(U)$$

M est une matrice de dimension JN. JN de terme général

$$M_{mn} = -Q^{-1} B^T \sum_{\ell=\text{Sup}(n+1, m+1)}^N A^{T(\ell-(n+1))} P A^{\ell-1-m} B, \quad \begin{matrix} n=0, \dots, N-1 \\ m=0, \dots, N-1 \end{matrix}$$

N est une matrice de dimension JN. I de terme général

$$N_n = -Q^{-1} B^T \sum_{\ell=n+1}^N A^{T(\ell-(n+1))} P A^{\ell}, \quad n=0, \dots, N-1$$

Définissons de plus l'application H

$$U = H(V) \quad \left\{ \begin{array}{ll} u_i^n = v_i^n & \text{si } u_{i \min} \leq v_i^n \leq u_{i \max} \\ u_i^n = u_{i \max} & \text{si } v_i^n > u_{i \max} \\ u_i^n = u_{i \min} & \text{si } v_i^n < u_{i \min} \end{array} \right.$$

Notons : $F = H \circ G$

Dans ces conditions les équations d'optimalité prennent la forme :

$$(3.11) \quad U = F(U)$$

La loi de commande optimale $\{\hat{u}^n\}$ est ainsi solution d'un système d'équations algébriques non linéaires (3.10) ou (3.11).

4 - CALCUL DE LA LOI DE COMMANDE OPTIMALE

La présence de contraintes ne permet pas d'utiliser une méthode classique de résolution d'un système linéaire. Nous allons décrire une méthode itérative simple à mettre en oeuvre, consistant à rechercher le point fixe de l'application F.

4.1. Technique de calcul

La suite $U_0, U_1, \dots, U_p$ définie par la relation de récurrence :

$$U_{p+1} = F(U_p) \quad (U_0 = \text{approximation initiale}) \text{ admet}$$

si elle converge une limite U point fixe de F .

Le calcul de U_{p+1} en fonction de U_p peut être effectué de deux manières différentes :

1 - En calculant directement V_{p+1} :

$$V_{p+1} = M U_p + N x_0$$

d'où l'on tire U_{p+1} en imposant les contraintes.

2 - En utilisant les relations (3.10)

Connaissant la loi de commande $\{u_p^n\}$ et $x_{p+1}^0 = x_0$ il est possible de déterminer la trajectoire $\{x_{p+1}^n\}$ par la récurrence :

$$x_{p+1}^{n+1} = A x_{p+1}^n + B u_p^n \quad n = 0, \dots, N-1$$

puis la trajectoire $\{\lambda_{p+1}^n\}$ par la récurrence inverse partant de $\lambda_{p+1}^N = 0$:

$$\lambda_{p+1}^{n-1} = P x_{p+1}^n + A^T \lambda_{p+1}^n \quad n = N, \dots, 1$$

et enfin v_{p+1}^n par l'équation :

$$v_{p+1}^n = - Q^{-1} B \lambda_{p+1}^n$$

d'où l'on tire comme précédemment u_{p+1}^n la nouvelle loi de commande.

Ces deux méthodes sont mathématiquement équivalentes le choix de l'une ou autre étant dicté par des impératifs techniques liés au calculateur numérique utilisé. La taille mémoire nécessaire, le nombre d'opérations à effectuer pour obtenir le résultat peuvent varier dans des proportions importantes suivant l'algorithme utilisé.

4.2. Convergence

4.2.1. Rappel [17]

Définition : L'application F de $\mathbb{R}^n$ dans $\mathbb{R}^n$ est contractante si il existe une constante $d \in [0,1]$ telle que pour tout $x, y \in \mathbb{R}^n$

$$\|F(x) - F(y)\| \leq d \|x - y\|$$

Théorème : Si F est contractante alors la suite définie par :

$$x^{n+1} = F(x) \quad n = 1, 2, 3, \dots x^0 \text{ quelconque est}$$

convergente. Sa limite est le point fixe unique de F .

4.2.2. Convergence du calcul de la loi de commande optimale

L'application au problème envisagé précédemment est immédiate : soit deux vecteurs quelconques de $\mathbb{R}^{JN}$ U et U' , pour chaque composante la relation suivante est vérifiée (Annexe I2) :

$$|F_j^n(U) - F_j^n(U')| \leq |v_j^n - v_j'^n|$$

d'où l'on tire immédiatement :

$$\|F(U) - F(U')\| \leq \|V - V'\| = \|M(U - U')\|$$

Soit la norme spectrale de M :

$$||M|| = \sup_{||U||=1} ||MU|| = \sqrt{\rho(M^T M)}$$

Il est alors possible d'écrire :

$$||F(U) - F(U')|| \leq ||M|| ||U - U'||$$

Une condition suffisante de convergence est que $d = ||M|| < 1$
ou encore $\rho(M^T M) < 1$.

Le calcul de $||M||$ pourra être effectué hors ligne par une méthode de recherche des valeurs propres. Si l'on cherche uniquement à s'assurer de la convergence une majoration de $||M||$ sera suffisante.

4.3. Paramètre de relaxation 18

Il est peut être utile (par exemple afin d'accroître la vitesse de convergence) d'introduire un paramètre de relaxation $\omega \in [0,2]$ définissant une nouvelle application F_ω ayant le même point fixe que F .

$$F_\omega(U) = \omega F(U) + (1 - \omega) U$$

$\omega < 1$ méthode de sous-relaxation

$\omega > 1$ méthode de sur-relaxation

Remarque : F se décompose naturellement en produit de deux applications

$$F = H \circ G$$

et il est alors possible de définir une autre famille d'applications F'_ω

$$F'_\omega = H \circ G_\omega$$

$$G_\omega(U) = \omega G(U) + (1 - \omega) U$$

admettant toujours le même point fixe que F (Annexe I-3).

5 - CONDUITE DU PROCESSUS

Il ne reste plus maintenant qu'à déterminer comment nous envisageons d'appliquer ces résultats à la conduite effective d'un processus industriel.

5.1. Conduite en boucle ouverte

La conduite en boucle ouverte du processus consiste à calculer hors ligne la loi de commande optimale puis à l'appliquer au processus sans tenir compte de l'évolution réelle de celui-ci. Ce type de conduite ne permet pas de tenir compte des imperfections du modèle et des perturbations imprévisibles agissant sur le processus.

5.2. Conduite en boucle fermée par intermittence

5.2.1. Fermeture de la boucle à chaque période

Ce type de conduite consiste, pendant chaque période d'échantillonnage, à calculer, en utilisant l'état acquis sur le processus, la commande qui sera appliquée au cours de la période suivante.

Le processus réel étant exactement régi par l'équation :

$$\underline{x}^{m+1} = A \underline{x}^m + B \underline{u}^m + \varepsilon^m$$

$$m = 0, 1, 2, \dots, \infty$$

on lui associe à l'instant m (début de l'horizon d'optimisation considéré) le modèle :

$$x^{n+1} = A x^n + B u^n,$$

$$\text{avec } x^0 = \underline{x}^m \quad n = 0, \dots, N-1$$

ε^m représente à la fois les perturbations et l'erreur sur le modèle

De plus nous admettrons que l'on peut déterminer à chaque instant $\underline{x}^m$ à l'aide de mesures effectuées sur le processus.

A partir de cette valeur il est facile d'obtenir, en utilisant le modèle, une prédiction de $\underline{x}^{m+1}$:

$$\underline{\hat{x}}^{m+1} = A \underline{x}^m + B \underline{u}^m$$

Il est alors possible de calculer entre les instants m et $m+1$ la commande optimale $\underline{u}^{m+1}$ définie comme :

$$\underline{u}^{m+1} \hat{=} \hat{u}^0 \text{ calculée pour } x^0 = \underline{\hat{x}}^{m+1}$$

La commande $\underline{u}^{m+1}$ sera ensuite appliquée entre les instants $m+1$ et

Cette procédure sera réitérée sans cesse assurant ainsi la conduite en boucle fermée du processus.

L'erreur commise sur $\underline{x}^{m+1}$ se réduit au terme ε^m traduisant l'erreur accumulée au cours de la seule période $m, m+1$.

A l'instant m l'horizon utilisé pour le calcul de $\underline{u}^m$ est l'horizon $[m, m+N]$, et à chaque nouveau calcul il sera décalé d'une période (horizon glissant).

5.2.2. Fermeture "intermittente" de la boucle

La méthode précédente est applicable uniquement si le calcul de la commande peut être effectué pendant une période d'échantillonnage. Dans le cas contraire la loi de commande optimale est calculée pendant k périodes d'échantillonnage puis appliquée pendant k périodes suivantes.

A l'instant m nous avons déterminé l'état x^m et nous allons calculer la loi de commande qui sera appliquée au processus entre les instants $m+k$ et $m+2k$. Comme précédemment nous estimons l'état à l'instant $m+k$:

$$\underline{x}^{m+k} = A^k \underline{x}^m + \sum_{\ell=m}^{m+k-1} A^{m+k-1-\ell} B \underline{u}^\ell$$

où $\{\underline{u}^m, \dots, \underline{u}^{m+k-1}\}$ sont les premières valeurs de la loi de commande calculée précédemment et appliquée au processus aux instants $m, \dots, m+k-1$.

Nous calculons ensuite la loi de commande optimale $\{\hat{u}^0, \dots, \hat{u}^{N-1}\}$ en utilisant à partir de $x^0 = \underline{x}^{m+k}$ le modèle :

$$x^{n+1} = A x^n + B u^n \quad n = 0, \dots, N-1$$

Nous ne conserverons que les premières valeurs de la loi de commande pour les appliquer au processus :

$$\underline{u}^{m+k} = \hat{u}^0, \dots, \underline{u}^{m+2k-1} = \hat{u}^{k-1}$$

Remarque :

Ce type de conduite est satisfaisant si la période de fermeture de la boucle (k périodes d'échantillonnage) est faible devant les constantes de temps du processus. Nous ne nous contentons pas, en effet, d'observer mais nous désirons commander aussi bien que possible un processus perturbé et représenté par un modèle approximatif. Il nous semble souvent préférable de modifier la période d'échantillonnage ou (et) différents paramètres, choisis pour définir le problème (horizon, critère) ou pour accélérer le calcul (paramètre de relaxation), afin de pouvoir fermer la boucle à chaque période.

CONCLUSION

Nous voulons conduire en boucle fermée un processus rapide en déterminant pendant chaque période d'échantillonnage une nouvelle valeur de la commande.

Nous allons voir comment il est possible d'effectuer en un temps limité les traitements itératifs nécessaires à l'aide de techniques de décomposition et par l'emploi de moyens de calcul rapides.

Nous proposerons également une méthode de conduite décentralisée avec coordination horizontale pour la commande de systèmes complexes constitués d'unités interconnectées.

ANNEXE I . 1

La matrice Q étant choisie diagonale le système (3.8) se décompose en J systèmes de la forme :

$$(A\ 1) \quad \left\{ \begin{array}{l} q_{ii} u_i^n + \sum_{j=1}^I b_{ji} \lambda_j^n = \mu_i^n (u_{i \max} + u_{i \min} - 2 u_i^n) \\ \mu_i^n (u_i^n - u_{i \max}) (u_i^n - u_{i \min}) = 0 \\ \text{avec :} \\ \mu_i^n \geq 0 \quad u_{i \min} \leq u_i^n \leq u_{i \max} \end{array} \right.$$

Il existe alors deux possibilités :

1) La solution u_i^n de l'équation :

$$(A\ 2) \quad q_{ii} u_i^n + \sum_{j=1}^I b_{ji} \lambda_j^n = 0 \quad (q_{ii} > 0)$$

appartient à l'intervalle fermé $[u_{i \min}, u_{i \max}]$; u_i^n est alors la solution, du système (A 1) obtenue pour $\mu_i^n = 0$.

Vérifions que le système (A 1) n'admet pas d'autre solution avec $\mu_i^n \neq 0$. Cette solution ne pourrait être alors que $u_{i \max}$ ou $u_{i \min}$:

a) $u_{i \max} \geq u_i^n$ et $q_{ii} > 0$ entraînent :

$$0 = q_{ii} u_i^n + \sum_{j=1}^I b_{ji} \lambda_j^n < q_{ii} u_{i \max} + \sum_{j=1}^I b_{ji} \lambda_j^n .$$

l'équation :

$$\underbrace{q_{ii} u_{i \max} + \sum_{j=1}^I b_{ji} \lambda_j^n}_{\geq 0} = \underbrace{\mu_i^n (u_{i \min} - u_{i \max})}_{< 0}$$

n'a donc pas de solution avec $\mu_i^n > 0$.

b) $u_{i \min} \leq u_i^n$, $q_{ii} > 0$ entraînent de même :

$$0 = q_{ii} u_i^n + \sum_{j=1}^I b_{ji} \lambda_j^n > q_{ii} u_{i \min} + \sum_{j=1}^I b_{ji} \lambda_j^n,$$

et comme précédemment l'équation :

$$\underbrace{q_{ii} u_{i \max} + \sum_{j=1}^I b_{ji} \lambda_j^n}_{\leq 0} = \underbrace{\mu_i^n (u_{i \max} - u_{i \min})}_{> 0}$$

n'a pas de solution avec $\mu_i^n > 0$

2) L'équation (A2) n'a pas de solution u_i^n appartenant à l'intervalle $[u_{i \min}, u_{i \max}]$; il n'existe donc pas de solution du système (A1) pour $\mu_i^n = 0$ et u_i^n prend donc l'une des valeurs $u_{i \max}$ ou $u_{i \min}$.

Notons v_i^n la solution de l'équation (A2) et envisageons les deux cas possibles :

a) $v_i^n > u_{i \max} > u_{i \min}$

La solution $u_i^n = u_{i \min}$ entraîne :

$$0 = q_{ii} v_i^n + \sum_{j=1}^I b_{ji} \lambda_j^n > q_{ii} u_{i \min} + \sum_{j=1}^I b_{ji} \lambda_j^n$$

$$\mu_i^n = \frac{q_{ii} u_{i \min} + \sum_{j=1}^I b_{ji} \lambda_j^n}{(u_{i \max} - u_{i \min})} < 0$$

ce qui est en contradiction avec l'hypothèse $\mu_i^n \geq 0$.

Au contraire la solution $u_i^n = u_{i \max}$ est correcte :

$$0 = q_{ii} v_i^n + \sum_{j=1}^I b_{ji} \lambda_j^n > q_{ii} u_{i \max} + \sum_{j=1}^I b_{ji} \lambda_j^n$$

$$\lambda_i^n = \frac{q_{ii} u_{i \max} + \sum_{j=1}^I b_{ji} \lambda_j^n}{(u_{i \min} - u_{i \max})} > 0$$

La solution est donc dans ce cas $u_i^n = u_{i \max}$.

$$b) \quad v_i^n < u_{i \min} < u_{i \max}$$

La solution $u_i^n = u_{i \max}$ implique :

$$0 = q_{ii} v_i^n + \sum_{j=1}^I b_{ji} \lambda_j^n < q_{ii} u_{i \max} + \sum_{j=1}^I b_{ji} \lambda_j^n$$

$$\mu_i^n = \frac{q_{ii} u_{i \max} + \sum_{j=1}^I b_{ji} \lambda_j^n}{(u_{i \min} - u_{i \max})} < 0$$

L'hypothèse $\mu_i^n \geq 0$ n'étant pas vérifiée nous éliminons cette solution. La solution $u_i^n = u_{i \min}$ convient :

$$0 = q_{ii} v_i^n + \sum_{j=1}^I b_{ji} \lambda_j^n < q_{ii} u_{i \min} + \sum_{j=1}^I b_{ji} \lambda_j^n$$

$$\mu_i^n = \frac{q_{ii} u_{i \min} - \sum_{j=1}^I b_{ji} \lambda_j^n}{(u_{i \max} - u_{i \min})} > 0$$

La solution est dans ce cas $u_i^n = u_{i \min}$

Nous obtenons donc le résultat (3.9)

ANNEXE I . 2

Nous allons démontrer que l'inéquation :

$$|F_j^n(U) - F_j^n(U')| \leq |v_j^n - v_j'^n|$$

est toujours vérifiée.

Pour alléger l'écriture notons :

$$v = v_j^n, \quad v' = v_j'^n \quad \text{et} \quad u = f_j^n(U), \quad u' = f_j^n(U').$$

Nous allons maintenant comparer

$$|u - u'| \quad \text{et} \quad |v - v'|$$

Pour cela étudions les différentes possibilités en tenant compte du rôle symétrique joué par v et v' :

$$1) \quad u_{\min} \leq v \leq u_{\max} \quad \text{et} \quad u_{\min} \leq v' \leq u_{\max}$$

$$\text{alors : } u = v, \quad u' = v' \quad \text{donc : } |u - u'| = |v - v'|$$

$$2) \quad u_{\min} \leq v \leq u_{\max} \quad \text{et} \quad v' \geq u_{\max}$$

$$\text{alors : } u = v, \quad u' = u_{\max} \quad \text{ce qui implique :}$$

$$0 \geq u - u' \geq v - v' \quad \text{et donc : } |u - u'| \leq |v - v'|$$

$$3) \quad u_{\min} \leq v \leq u_{\max} \quad \text{et} \quad v' \leq u_{\min}$$

$$\text{alors : } u = v, \quad u' = u_{\min} \quad \text{ce qui implique :}$$

$$0 \leq u - u' \leq v - v' \quad \text{et donc : } |u - u'| \leq |v - v'|, \quad ,$$

$$4) \quad v \geq u_{\max} \quad \text{et} \quad v' \geq u_{\max}$$

$$\text{alors :} \quad u = u_{\max} = u' \quad \text{donc :} \quad |u - u'| = 0 \leq |v - v'|$$

$$5) \quad v \geq u_{\max} \quad \text{et} \quad v' \leq u_{\min}$$

$$\text{alors :} \quad u = u_{\max}, u' = u_{\min} \quad \text{ce qui entraîne :}$$

$$0 \leq u - u' \leq v - v' \quad \text{et donc :} \quad |u - u'| \leq |v - v'|$$

$$6) \quad v \leq u_{\min} \quad \text{et} \quad v' \leq u_{\min}$$

$$\text{alors :} \quad u = u_{\min}, u' = u_{\min} \quad \text{donc :} \quad |u - u'| = 0 \leq |v - v'|$$

Dans tous les cas nous avons donc :

$$|u - u'| \leq |v - v'|$$

ANNEXE I.3

Soit l'application F :

$$F = H \circ G$$

et l'application F'_ω :

$$\left\{ \begin{array}{l} F'_\omega = H \circ G_\omega \\ G_\omega = \omega G(U) + (1 - \omega) U \end{array} \right. \quad \omega > 0$$

Nous allons montrer qu'elles admettent le(s) même(s) point(s) fixe(s).

Pour alléger les notations nous posons :

$u = u_i^n$ une composante du vecteur U

$v = v_i^n$ une composante du vecteur V

et :

$f = f_i^n$ une composante de la fonction F

$g = g_i^n$ une composante de la fonction G

1 - Supposons que F'_ω admette U comme point fixe. Pour chaque composante scalaire nous avons les relations :

$$\left\{ \begin{array}{ll} v = \omega g(u) + (1 - \omega) u & \omega > 0 \\ \begin{array}{ll} u = v & \text{si} \quad u_{\min} \leq v \leq u_{\max} \\ u = u_{\max} & \text{si} \quad v > u_{\max} \\ u = u_{\min} & \text{si} \quad v < u_{\min} \end{array} \end{array} \right.$$

Etudions les différentes possibilités :

$$u_{\min} \leq v \leq u_{\max}$$

alors $u = v$ ce qui entraîne :

$$u = \omega g(U) + (1-\omega) u$$

$$g(U) = u \text{ et enfin : } f(U) = u$$

$$v > u_{\max}$$

alors $u = u_{\max}$ ce qui implique :

$$v = g(U) + (1-\omega) u_{\max}$$

$$g(U) = u_{\max} + \frac{1}{\omega} (v - u_{\max})$$

$$g(U) > u_{\max} \text{ et donc : } f(U) = u$$

$$v < u_{\min}$$

alors $u = u_{\min}$ ce qui entraîne :

$$v = \omega g(U) + (1-\omega) u_{\min}$$

$$g(U) = u_{\min} + \frac{1}{\omega} (v - u_{\min})$$

$$g(U) < u_{\min} \text{ et par conséquent : } f(U) = u$$

Cela restant vrai pour toute composante U est donc point fixe de F :

$$U = F(U)$$

2 - Supposons que F admette U comme point fixe. Pour chaque composante scalaire nous pouvons écrire :

$$\left\{ \begin{array}{lll} v = g(U) & & \\ & u = v & \text{si } u_{\min} \leq v \leq u_{\max} \\ & u = u_{\max} & \text{si } v > u_{\max} \\ & u = u_{\min} & \text{si } v < u_{\min} \end{array} \right.$$

Etudions les différentes possibilités :

$$u_{\min} \leq v \leq u_{\max}$$

$u = g(U)$ et en multipliant chaque membre par ω ($\omega > 0$) :

$$u = \omega g(U) + (1-\omega)u = g'_{\omega}(U)$$

$$u_{\min} \leq g'_{\omega}(U) \leq u_{\max} \text{ et donc : } f'_{\omega}(U) = u$$

$$v > u_{\max}$$

$$u = u_{\max}$$

$$g'_{\omega}(U) = \omega [g(U) - u_{\max}] + u_{\max}$$

$$g'_{\omega}(U) = \omega [v - u_{\max}] + u_{\max}$$

or nous avons : $\omega > 0$ et $v > u_{\max}$

et $g'_{\omega}(U) > u_{\max}$ et par conséquent : $f'_{\omega}(U) = u$

$$v < u_{\min}$$

$$u = u_{\min}$$

$$g'_{\omega}(U) = \omega [g(U) - u_{\min}] + u_{\min}$$

$$g'_{\omega}(U) = \omega [v - u_{\min}] + u_{\min}$$

$$\text{or puisque : } \omega > 0 \text{ et } v < u_{\min}$$

$$g'_{\omega}(U) < u_{\min} \text{ et enfin : } f'_{\omega}(U) = u$$

L'égalité restant vérifiée pour chaque composante U est donc également le point fixe de F'_{ω}

$$U = F'_{\omega}(U)$$

Nous avons donc montré que tout point fixe de F'_{ω} est point fixe de F et que, réciproquement tout point fixe de F est point fixe de F'_{ω} .

II - DECOMPOSITION DES CALCULS ET COMMANDE DECENTRALISEE AVEC COORDINATION

1 - SYSTEMES INTERCONNECTES

- 1.1. Le modèle
- 1.2. Le critère
- 1.3. Exemples de systèmes interconnectés

2 - DECOMPOSITION DES CALCULS

- 2.1. Recherche du point fixe d'une application par un
algorithme de relaxation chaotique à retards
 - 2.1.1. Itérations chaotiques
 - 2.1.2. Itérations chaotiques à retards
 - 2.1.3. Convergence
- 2.2. Application au calcul de la loi de commande optimale
 - 2.2.1. Décomposition de l'application F
 - 2.2.2. Décomposition des équations d'optimalité
 - 2.2.3. Problèmes locaux et coordination
 - 2.2.4. Calcul hiérarchisé et décomposition horizontale

3 - COMMANDE DECENTRALISEE

- 3.1. Méthode optimale
- 3.2. Méthodes sous-optimales
 - 3.2.1. Coordination statique
 - 3.2.2. Coordination dynamique
 - 3.2.2.1. Fonctionnement synchrone
 - 3.2.2.2. Fonctionnement asynchrone
- 3.3. Méthode mixte

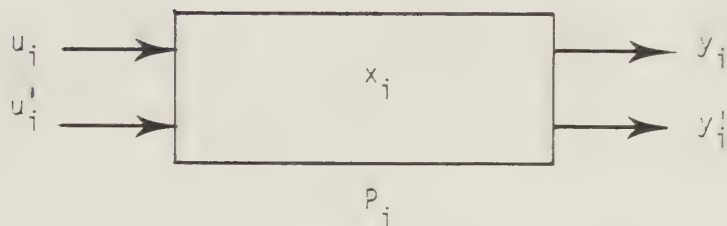
CONCLUSION

Nous allons traiter dans ce chapitre le problème de la commande optimale d'un système linéaire de grande taille composé de sous-processus interconnectés. Nous étudierons la décomposition du calcul de la loi de commande optimale puis nous indiquerons des méthodes de conduite décentralisée du processus.

1 - SYSTEMES INTERCONNECTES

1.1. Le modèle

Considérons un processus quelconque P_i ayant pour entrées les vecteurs u_i et u'_i et pour sortie les vecteurs y_i et y'_i



Construisons un processus P en connectant I processus P_i de la manière suivante :

- les entrées u_i forment la commande de P :

$$u = \begin{bmatrix} u_1 \\ \vdots \\ u_I \end{bmatrix}$$

- les sorties y_i forment la sortie de P :

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_I \end{bmatrix}$$

- la connexion entre les processus P_i est assurée en formant chaque entrée u'_i à partir des sorties y_j , $j = i$:

$$u'_i = \begin{bmatrix} y'_1 \\ \vdots \\ y'_{i-1} \\ y'_{i+1} \\ \vdots \\ y'_I \end{bmatrix}$$

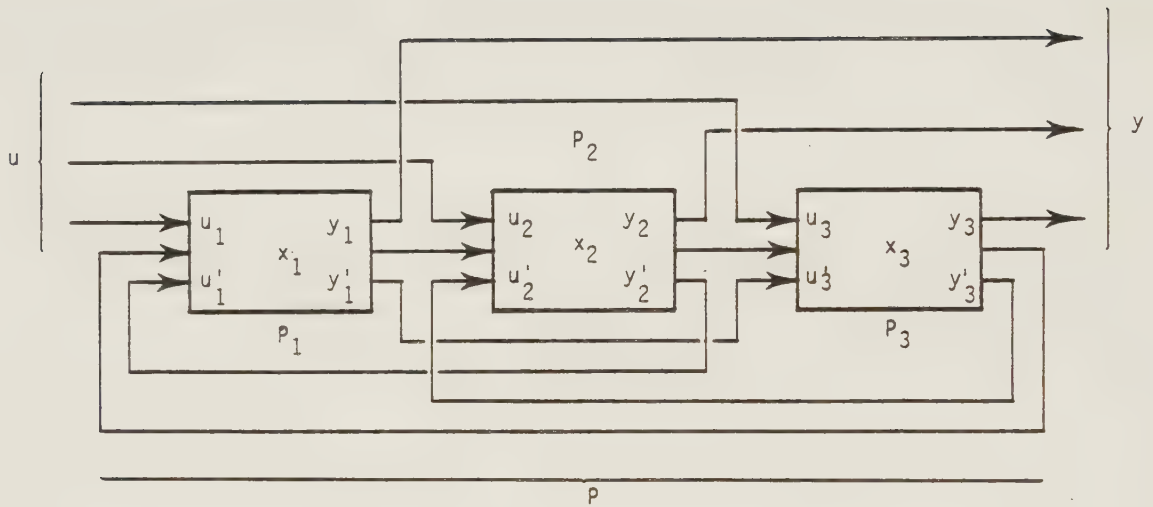


Figure 1 : Interconnexion de 3 processus

Nous allons former le modèle du processus global P à partir des modèles des sous-processus P_i :

$$\begin{cases} x_i^{n+1} = A_{ii} x_i^n + (B_{ii}, B'_i) (u_i^n, u_i'^n)^T, x_i^0 \text{ fixé}, n = 0, \dots, N-1 \\ (y_i^n, y_i'^n)^T = (C_{ii}, C'_i)^T x_i^n \end{cases}$$

Posons :

$$B'_i = (B_{i1}, \dots, B_{i \ i-1}, B_{i \ i+1}, \dots, B_{iI})$$

telle que :

$$B'_i u_i^n = \sum_{j \neq i} B_{ij} y_j^n = \sum_{j \neq i} B_{ij} C'_j x_j^n$$

Le modèle de P_i prend alors la forme :

$$(1-1) \left\{ \begin{array}{l} x_i^{n+1} = A_{ii} x_i^n + B_{ii} u_i^n + \sum_{j \neq i} A_{ij} x_j^n, \ x_i^0 \text{ fixé}, \ n = 0, \dots, N-1 \\ y_i = C_{ii} x_i^n \end{array} \right.$$

avec :

$$A_{ij} = B_{ij} C'_j$$

Le modèle du processus global P s'écrit :

$$(1-2) \left\{ \begin{array}{l} x^{n+1} = A x^n + B u^n \quad x^0 \text{ fixé}, \ n = 0, \dots, N-1 \\ z^n = C x^n \end{array} \right.$$

avec :

$$A = \begin{bmatrix} A_{11} & \dots & A_{1I} \\ \vdots & & \vdots \\ A_{I1} & \dots & A_{II} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & & 0 \\ & \ddots & \\ 0 & & B_{II} \end{bmatrix}, \quad C = \begin{bmatrix} C_{11} & & 0 \\ & \ddots & \\ 0 & & C_{II} \end{bmatrix}$$

Réciproquement tout processus ayant un modèle du type (1-2) peut être considéré comme composé de sous-processus interconnectés de la manière décrite précédemment.

1.2. Le critère

A ce modèle est associé un critère quadratique "additif séparable" :

$$J = \sum_{i=1}^I J_i$$

où J_i est un critère local :

$$(1-3) \quad J_i = \sum_{n=1}^N \left(\begin{bmatrix} x_i^n \end{bmatrix}^T P_{ii} x_i^n + \begin{bmatrix} u_i^{n-1} \end{bmatrix}^T Q_{ii} u_i^{n-1} \right)$$

Le critère global J prend alors la forme :

$$(1-4) \quad J = \sum_{n=1}^N \left(\begin{bmatrix} x^n \end{bmatrix}^T P x^n + \begin{bmatrix} u^{n-1} \end{bmatrix}^T Q u^{n-1} \right)$$

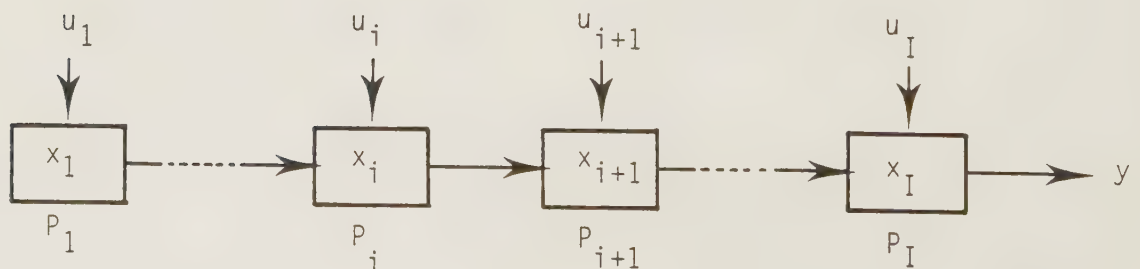
avec :

$$P = \begin{bmatrix} P_{11} & & 0 \\ & \ddots & \\ 0 & & P_{II} \end{bmatrix}, \quad Q = \begin{bmatrix} Q_{11} & & 0 \\ & \ddots & \\ 0 & & Q_{II} \end{bmatrix}$$

1.3. Exemples de systèmes interconnectés

1.3.1. Système en cascade

Les sous-processus P_i sont connectés en série, la sortie du processus P_i étant utilisée comme entrée pour le processus P_{i+1}

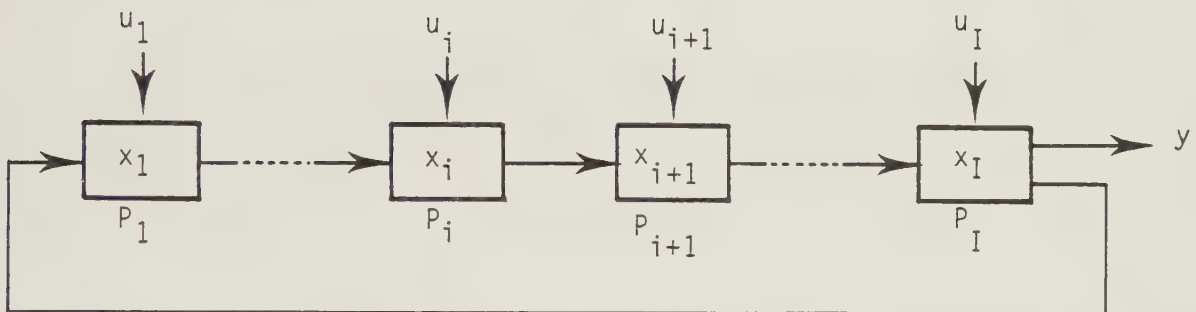


Les matrices A, B et C se réduisent à :

$$A = \begin{bmatrix} A_{11} & & & 0 \\ A_{21} & & & \\ & \ddots & & \\ 0 & A_{I \ I-1} & & A_{II} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & & 0 \\ & \ddots & \\ 0 & & B_{II} \end{bmatrix} \quad C = \begin{bmatrix} 0 & & 0 \\ & \ddots & \\ 0 & & C_{II} \end{bmatrix}$$

1.3.2. Système en anneau

Ce système est obtenu en ajoutant au précédent une réaction de la sortie de P_I sur l'entrée de P_1 .



Les matrices A, B et C s'écrivent alors :

$$A = \begin{bmatrix} A_{11} & & & A_{II} \\ A_{21} & & & 0 \\ & \ddots & & \\ 0 & A_{I \ I-1} & & A_{II} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & & 0 \\ & \ddots & \\ 0 & & B_{II} \end{bmatrix} \quad C = \begin{bmatrix} 0 & & 0 \\ & \ddots & \\ 0 & & C_{II} \end{bmatrix}$$

De façon plus générale nous considérerons des systèmes possédant un nombre limité d'interconnexions, c'est-à-dire représentés par une matrice d'état A largement creuse en dehors des blocs A_{ij} de sa diagonale (Ceci ne préjuge pas de l'intensité des interactions existantes).

2 - DECOMPOSITION DU CALCUL

Nous nous proposons, maintenant, de décomposer le calcul de la loi de commande optimale, par exemple pour l'effectuer sur un calculateur multiprocesseur. Nous utiliserons une décomposition horizontale du problème correspondant à la structure du processus. La coordination des sous-problèmes sera effectuée suivant un algorithme permettant un fonctionnement asynchrone des éléments de calcul.

2.1. Recherche du point fixe d'une application par un algorithme de relaxation chaotique à retards (Rappel [16])

Nous recherchons le point fixe x d'une application quelconque F

$$(2-1) \quad x = F(x).$$

Nous décomposons le vecteur x en I sous-vecteurs x_i :

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_I \end{bmatrix}$$

De même l'application F se met sous la forme :

$$F(x) = \begin{bmatrix} F_1(x) \\ \vdots \\ F_I(x) \end{bmatrix}$$

L'équation (2-1) devient alors un système :

$$(2-2) \quad x_i = F_i(x) \quad i = 1, \dots, I$$

La solution peut être obtenue par relaxation en traitant en parallèle et de manière éventuellement asynchrone les équations (2-2).

2.1.1. Itérations chaotiques

Une suite $\{s^p\}$, $p = 0, 1, \dots$ est une suite chaotique si

$$(2.3) \quad \forall p \quad s^p \subset \{1, 2, \dots, I\} \quad , \quad s^p \neq \emptyset$$

l'ensemble $\{p/i \in s^p\}$ est infini, $i = 1, \dots, I$

La suite $\{s^p\}$ définit un algorithme chaotique :

$$(2.4) \quad \begin{cases} x_i^{p+1} = F_i(x^p) & \text{si } i \in s^p \\ x_i^{p+1} = x_i^p & \text{si } i \notin s^p \end{cases} \quad x^0 \text{ fixé, } p = 0, 1, \dots$$

Le calcul est effectué séquentiellement pour :

$$s^p = \{1 + (p) \text{ modulo } I\} \quad (\text{Méthode de Gauss-Seidel})$$

ou au contraire en parallèle en prenant :

$$s^p = \{1, \dots, I\} \quad (\text{Méthode de Jacobi})$$

ou encore suivant un algorithme de type "série-parallèle". Cette dernière méthode de calcul correspond au fonctionnement synchrone d'un calculateur multiprocesseur.

2.1.2. Itérations chaotiques à retards

A la suite $\{s^p\}$ définie précédemment nous ajoutons une suite de vecteurs de retards $r^p = (r_1^p, \dots, r_I^p)^T$ à composantes entières telle qu'il existe un entier k et une suite k^p vérifiant :

$$(2.5) \quad \begin{cases} p - k^p \text{ non décroissante} \\ k^p \leq \min(k, p) \\ 0 \leq r_i^p \leq k^p \end{cases} \quad i = 0, \dots, I \quad , \quad p = 0, 1, \dots$$

On forme ainsi une suite de retards bornés et progressifs. A la double suite $\{s^p, r^p\}$ est associé un algorithme de relaxation chaotique à retards :

$$(2.6) \quad \begin{cases} x_i^{m+1} = F_i(x_1^{p-r_1^p}, \dots, x_I^{p-r_I^p}) & \text{si } i \in s^p \\ x_i^{m+1} = x_i^m & \text{si } i \notin s^p \end{cases} \quad , \quad x^0 \text{ fixé, } p = 0, 1, \dots$$

Cet algorithme, incluant le précédent, permet de plus de traduire le fonctionnement asynchrone d'un calculateur multiprocesseur.

2.1.4. Convergence

On appelle norme vectorielle canonique du vecteur x , pour la décomposition considérée, le vecteur de $\mathbb{R}_+^I$:

$$|||x||| = (||x_1||, \dots, ||x_I||)^T$$

Une application F est contractante pour la norme vectorielle canonique s'il existe une matrice D de dimension $I.I$ à coefficients non négatifs telle que pour tout (x,y) :

$$(2.7) \quad |||F(x) - F(y)||| < D \quad |||x - y|||, \rho(D) < 1$$

Théorème : Si une application F est contractante pour la norme vectorielle canonique alors la suite définie en (2-6) converge vers le point fixe unique de F .

2.2. Application au calcul de la loi de commande optimale

2.2.1. Décomposition de l'application F

Reprenons les résultats obtenus au chapitre I (3-11) : la loi de commande optimale est le point fixe de l'application F :

$$U = F(U) \qquad F = H \circ G$$

$$V = G(U) = MU + NX_0 \qquad H \text{ assure le respect des contraintes}$$

Cette équation peut se décomposer arbitrairement en un système de I équations dépendantes que l'on peut résoudre par un algorithme de relaxation chaotique à retards :

$$U_i = F_i(U)$$

c'est-à-dire :

$$(2.8) \quad \begin{cases} V_i = \sum_{j=1}^I M_{ij} U_j + N_i X_0 & i = 1, \dots, I \\ U_i = H_i(V_i) \end{cases}$$

Nous allons rechercher une matrice de contraction pour F suivant une méthode analogue à celle utilisée au chapitre I.

Pour chaque sous-calcul nous avons pour tout (U, U') :

$$||F_i(U) - F_i(U')|| < || \sum_{j=1}^I M_{ij} U_j - \sum_{j=1}^I M_{ij} U'_j ||$$

$$||F_i(U) - F_i(U')|| < \sum_{j=1}^I ||M_{ij}|| \cdot ||U_j - U'_j||$$

Pour le problème global nous obtenons :

$$|||F(U) - F(U')||| < D |||U - U'|||$$

avec :

$$D = \begin{bmatrix} ||M_{11}|| & ||M_{1I}|| \\ ||M_{I1}|| & ||M_{II}|| \end{bmatrix}$$

D est la matrice de contraction si $\rho(D) < 1$.

Pour qu'il soit possible d'obtenir la solution du problème en utilisant un algorithme de relaxation chaotique à retard il est suffisant que la matrice $[||M_{ij}||]$ ait son rayon spectral inférieur à l'unité.

Cette méthode de calcul où la décomposition est arbitraire reflète mal la structure du processus et ne permet pas en particulier de tenir compte de l'aspect "systèmes interconnectés".

2.2.2. Décomposition des équations d'optimalité

Nous considérons un ensemble de processus interconnectés et nous recherchons une méthode de décomposition, du calcul de la loi de commande optimale, correspondant à la structure du système.

Les équations d'optimalité du processus global ayant un modèle du type (1-2) pour un critère du type (1-3) s'écrivent :

$$(2.9) \left\{ \begin{array}{l} x^{n+1} = Ax^n + Bu^n \\ \lambda^{n-1} = Px^n + A^T \lambda^n \\ v^n = -Q^{-1} B^T \lambda^n \end{array} \right. \quad \begin{array}{l} x^0 \text{ fixé} , \quad n = 0, \dots, N-1 \\ \lambda^N = 0 \quad , \quad n = N, \dots, 1 \end{array}$$

$$\begin{array}{l} u_k^n = v_k^n \quad \text{si} \quad u_{k \min} \leq v_k^n \leq u_{k \max} \\ u_k^n = u_{k \max} \quad \text{si} \quad v_k^n > u_{k \max} \\ u_k^n = u_{k \min} \quad \text{si} \quad v_k^n < u_{k \min} \end{array}$$

(u_k est une composante scalaire de u)

Nous décomposons le système (2-9) en I sous-systèmes en respectant l'architecture du processus :

$$(2.10) \left\{ \begin{array}{l} x_i^{n+1} = \sum_{j=1}^I A_{ij} x_j^n + B_{ii} u_i^n \\ \lambda_i^{n-1} = P_{ii} x_i^n + \sum_{j=1}^I A_{ji}^T \lambda_j^n \\ v_i^n = -Q_{ii}^{-1} B_{ii}^T \lambda_i^n \end{array} \right. \quad \begin{array}{l} x_i^0 \text{ fixé} , \quad n = 0, \dots, N-1 \\ \lambda_i^N = 0 \quad , \quad n = N, \dots, 1 \end{array}$$

$$\begin{array}{l} u_{ik}^n = v_{ik}^n \quad \text{si} \quad u_{ik \min} \leq v_{ik}^n \leq u_{ik \max} \\ u_{ik}^n = u_{ik \max} \quad \text{si} \quad v_{ik}^n > u_{ik \max} \\ u_{ik}^n = u_{ik \min} \quad \text{si} \quad v_{ik}^n < u_{ik \min} \end{array}$$

(u_{ik} est une composante du vecteur u_i)

Nous allons calculer la loi de commande optimale du processus global en utilisant un algorithme de relaxation chaotique à retards.

Chaque calcul local consiste à obtenir les trajectoires x_i^n , λ_i^n et u_i^n en fonction d'une trajectoire u_i^n précédente, en procédant de la manière suivante :

- à partir de la loi de commande u_i^n on calcule la trajectoire d'état x_i^n en utilisant l'équation de récurrence directe
- on établit la trajectoire adjointe λ_i^n , en fonction de la trajectoire x_i^n calculée précédemment, par l'équation de récurrence inverse
- enfin on obtient la nouvelle loi de commande u_i^n , à partir de la trajectoire adjointe que l'on vient de calculer λ_i^n , en appliquant les équations non linéaires.

Posons :

$$\begin{aligned} X_i &= (x_i^1, \dots, x_i^N)^T & X &= (x_1, \dots, x_I)^T \\ \Lambda_i &= (\lambda_i^0, \dots, \lambda_i^{N-1})^T & \Lambda &= (\Lambda_1, \dots, \Lambda_I)^T \\ V_i &= (v_i^0, \dots, v_i^{N-1})^T & V &= (V_1, \dots, V_I)^T \\ U_i &= (u_i^0, \dots, u_i^{N-1})^T & U &= (U_1, \dots, U_I)^T \end{aligned}$$

Chaque sous-système d'équation (2-10) est équivalent, pour la méthode de calcul envisagée, à l'équation :

$$(2.11) \quad \begin{bmatrix} X_i \\ \Lambda_i \\ V_i \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & K_{ii} \\ 0 & 0 & L_{ii} \\ 0 & 0 & M_{ii} \end{bmatrix}}_{\Delta_{ii}} \cdot \begin{bmatrix} X_i \\ \Lambda_i \\ U_i \end{bmatrix} + \sum_{j \neq i} \underbrace{\begin{bmatrix} K_{ij} & 0 & 0 \\ L_{ij} & L'_{ij} & 0 \\ M_{ij} & M'_{ij} & 0 \end{bmatrix}}_{\Delta_{ij}} \cdot \begin{bmatrix} X_j \\ \Lambda_j \\ U_j \end{bmatrix} + N X^0$$

à laquelle nous ajoutons les équations assurant le respect des contraintes que nous noterons

$$U_i = H_i(V_i)$$

Posons pour simplifier l'écriture :

$$\begin{aligned}
 W_i &= (X_i, \Lambda_i, U_i)^T \\
 W &= (W_1, \dots, W_I)^T
 \end{aligned}
 , \quad
 \Delta = \begin{bmatrix} \Delta_{11} & \dots & \Delta_{1I} \\ \vdots & & \\ \Delta_{I1} & \dots & \Delta_{II} \end{bmatrix}$$

et notons l'équation (2.11) :

$$W_i = F'_i(W)$$

Prenons deux vecteurs W et W' quelconques ; nous pouvons écrire :

$$||H_i(V_i) - H_i(V'_i)|| < ||V_i - V'_i||$$

Nous obtenons immédiatement l'inégalité :

$$\begin{aligned}
 (2.12) \quad ||F'_i(W) - F'_i(W')|| &< ||\sum_{j=1}^I \Delta_{ij} W_j - \sum_{j=1}^I \Delta_{ij} W'_j|| \\
 ||F'_i(W) - F'_i(W')|| &< \sum_{j=1}^I ||\Delta_{ij}|| \cdot ||W_j - W'_j||
 \end{aligned}$$

Nous recherchons le point fixe de l'application F définie comme suit :

$$F'(W) = (F'_1(W), \dots, F'_I(W))^T$$

En concaténant les inégalités (2.12) nous avons :

$$|||F'(W) - F'(W')||| < D' |||W - W'|||$$

$$D' = \begin{bmatrix} ||\Delta_{11}|| & ||\Delta_{1I}|| \\ ||\Delta_{I1}|| & ||\Delta_{II}|| \end{bmatrix}$$

La condition suffisante de convergence d'un algorithme de relaxation chaotique à retards est :

$$\rho(D') < 1$$

2.2.3. Problèmes locaux et coordination

L'algorithme de relaxation chaotique à retards permet d'effectuer les calculs dans un ordre quelconque. Pour l'utilisation d'un système multiprocesseur il est intéressant de limiter les échanges entre les unités de calcul. Nous allons donc réitérer le plus longtemps possible chaque calcul local, indépendamment des autres, c'est-à-dire jusqu'à convergence vers une solution locale.

Nous mettons ainsi en évidence un problème de commande optimale local dont les équations sont :

$$(2.13) \left\{ \begin{array}{ll} x_i^{n+1} = A_{ii} x_i^n + B_{ii} u_i^n + e_i^n & x_i^0 \text{ fixé, } n = 0, \dots, N \\ \lambda_i^{n-1} = P_{ii} x_i^n + A_{ii}^T \lambda_i^n + f_i^n & \lambda_i^N = 0, n = N, \dots, 1 \\ v_i^n = -Q_{ii}^{-1} B_{ii}^T \lambda_i^n \\ \\ u_{ik}^n = v_{ik}^n & \text{si } u_{ik}^{\min} < v_{ik}^n < u_{ik}^{\max} \\ u_{ik}^n = u_{ik}^{\max} & \text{si } v_{ik}^n > u_{ik}^{\max} \\ u_{ik}^n = u_{ik}^{\min} & \text{si } v_{ik}^n < u_{ik}^{\min} \end{array} \right.$$

e_i^n et f_i^n sont des trajectoires de couplage :

$$e_i^n = \sum_{j \neq i} A_{ij} x_j^n, \quad f_i^n = \sum_{j \neq i} A_{ji}^T \lambda_j^n$$

Nous reconnaissons le problème de la commande optimale d'un processus P_i soumis à une perturbation e_i^n :

$$x_i^{n+1} = A_{ii} x_i^n + B_{ii} u_i^n + e_i^n$$

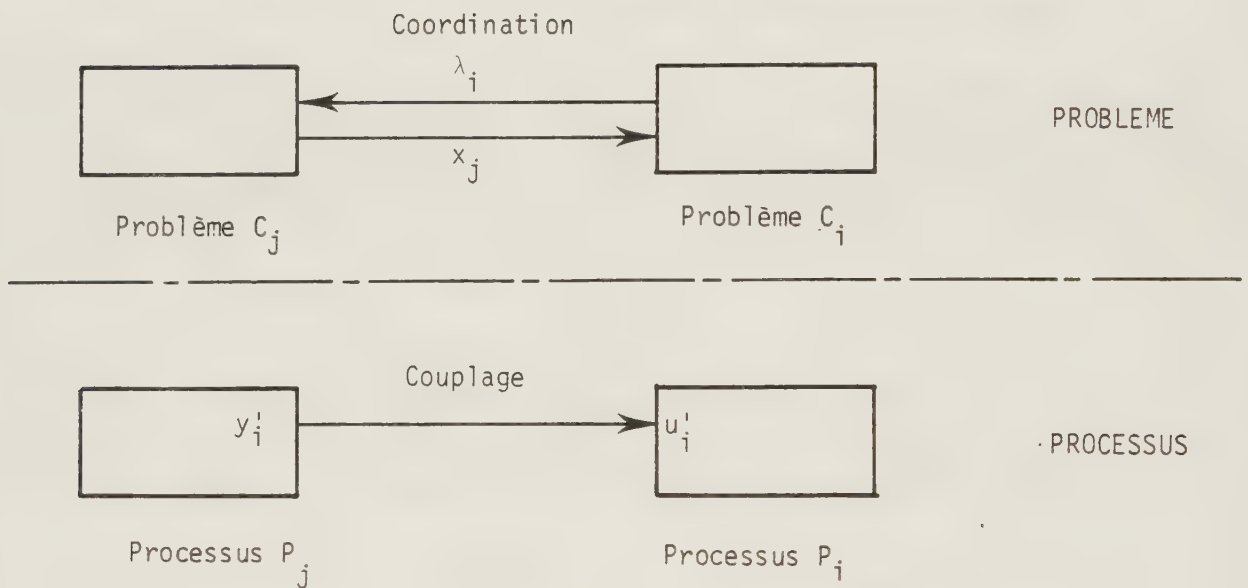
pour un critère du type :

$$J_i' = \sum_{n=1}^N \left([x_i^n]^T P_{ii} x_i^n + [u_i^{n-1}]^T Q_{ii} u_i^{n-1} + [f_i^n]^T x_i^n \right)$$

obtenu par adjonction d'un terme linéaire en x_i^n au critère local :

$$J_i' = J_i + \sum_{n=1}^N [f_i^n]^T x_i^n$$

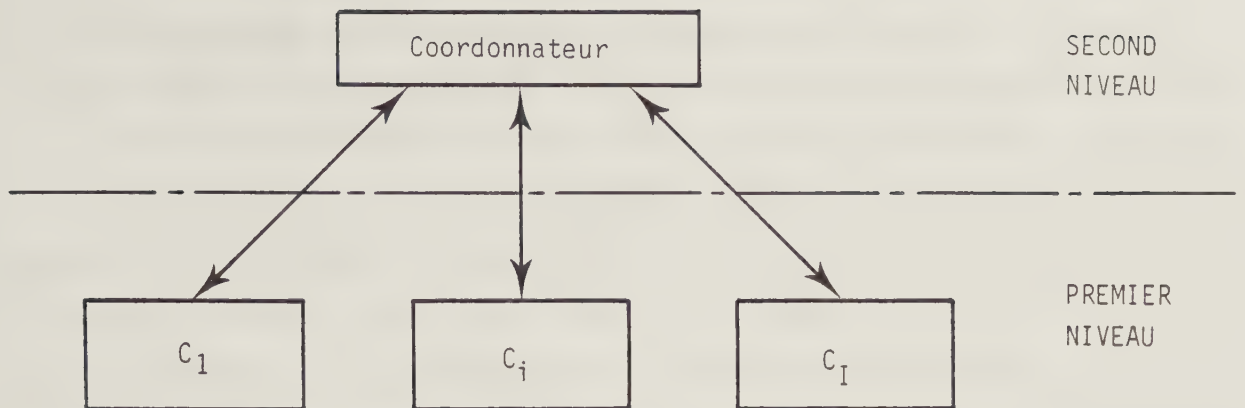
La coordination entre les sous-problèmes est effectuée en agissant d'une part sur le modèle par le terme e_j d'autre part sur le critère par le terme f_j . Les échanges d'informations entre les sous-problèmes, nécessaires uniquement lorsqu'il existe des couplages entre les processus associés, prennent la forme suivante :



2.2.4. Calcul hiérarchisé et décomposition horizontale

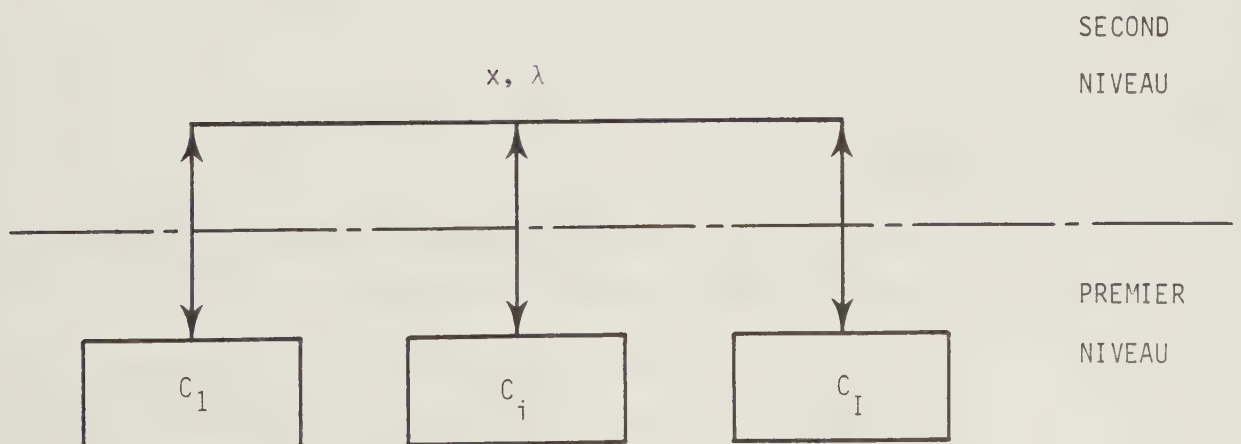
Les méthodes classiques de calcul hiérarchisé, dérivées de la théorie de la commande hiérarchisée à deux niveaux, consistent à résoudre le problème de la manière suivante [11] :

- Un premier niveau est chargé du calcul de la loi de commande optimale pour chaque sous-processus en utilisant un modèle et un critère locaux
- Un deuxième niveau intervient sur les modèles et (ou) les critères en vue de réaliser l'optimisation globale.



Notre approche diffère par deux points essentiels :

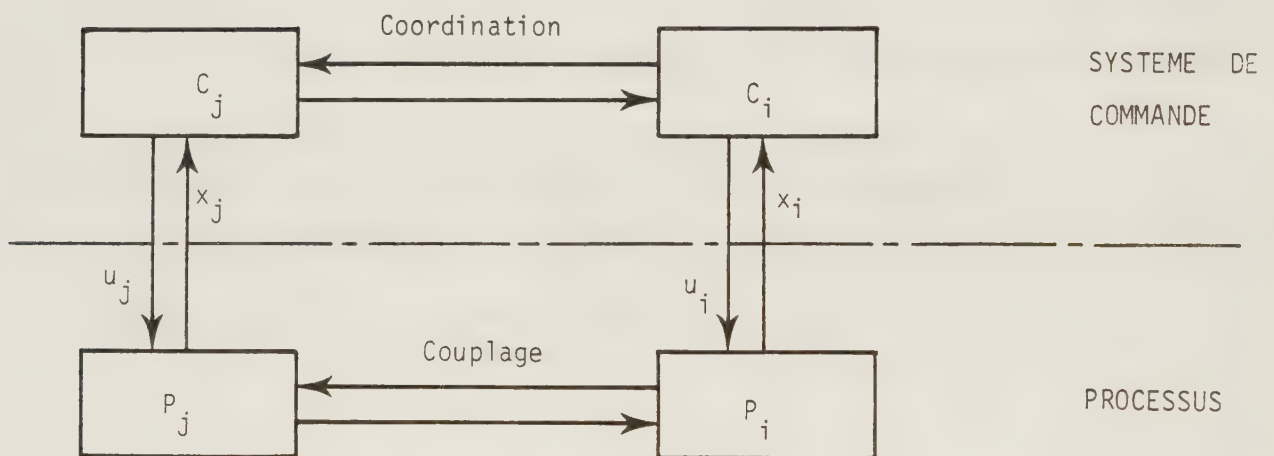
- Nous cherchons à calculer la solution globale du problème, la décomposition en sous-problèmes est uniquement la conséquence d'une configuration particulière de l'algorithme utilisé.
- Le second niveau est inexistant ou plus exactement se réduit à l'échange d'informations entre les sous-problèmes.



3 - COMMANDE DECENTRALISEE AVEC COORDINATION

Les processus industriels complexes sont souvent composés d'éléments répartis sur le site. D'autre part il est nécessaire de pouvoir implanter progressivement le système de commande.

Nous allons donc envisager la commande décentralisée de processus interconnectés. Chaque processus étant muni de son propre système de commande, nous allons décrire des méthodes permettant de coordonner les actions locales.



3.1. Méthode "optimale"

Nous allons conduire le processus global en boucle fermée de la manière décrite au chapitre précédent.

A partir de valeurs acquises sur le processus à un instant n on calcule la commande à appliquer à l'instant $n+1$. Ce calcul est réparti sur les différents systèmes de commande et effectué suivant la méthode décrite au paragraphe précédent. La détermination de l'état du processus et l'émission de la commande sont

réalisés au niveau local.

Tout en permettant de conserver une vision globale du processus cette méthode permet une introduction progressive des systèmes de commande et de la coordination entre ceux-ci. En revanche la méthode de calcul itérative employée demande un nombre important d'échanges pour obtenir la solution globale du problème. Le calcul ne pourra pas toujours être effectué en un temps suffisamment court ; nous allons donc décrire des méthodes de commande sous-optimales éventuellement plus aptes à la conduite en ligne.

3.2. Méthodes sous-optimales

Nous n'effectuons plus le calcul global de la loi de commande optimale mais nous résolvons des problèmes locaux coordonnés de façon approximative.

3.2.1. Coordination statique

Nous considérons les termes de couplage comme étant constants ; nous nous trouvons alors en présence de problèmes de commande optimale dont les équations d'optimalité sont :

$$(3.1) \quad \begin{cases} x_i^{n+1} = A_{ii} x_i^n + B_{ii} u_i^n + e_i & x_i^0 \text{ fixé, } n=0, \dots, N-1 \\ \lambda_i^{n-1} + P_{ii} x_i^n + A_{ii}^T \lambda_i^n + f_i & \lambda_i^N = 0, \quad n=N, \dots, 1 \end{cases}$$

la troisième équation restant inchangée.

Les paramètres de couplage e_i et f_i pouvant être calculés en utilisant par exemple les valeurs moyennes $\bar{x}_j$ et $\bar{\lambda}_j$ des trajectoires d'état et des trajectoires adjointes obtenues lors d'un calcul précédent :

$$e_i = \sum_{j \neq i} A_{ij} \bar{x}_j, \quad f_i = \sum_{j \neq i} A_{ji}^T \bar{\lambda}_j$$

La faible quantité d'informations à transmettre entre les systèmes de commande est le principal avantage de cette méthode.

3.2.2. Coordination dynamique

Le calcul de la loi de commande est maintenant réalisé pour chaque sous-système à partir d'une estimation, effectuée précédemment, de l'évolution des autres processus. Les systèmes de commande peuvent fonctionner en mode synchrone ou asynchrone.

3.2.2.1. Fonctionnement synchrone

Tous les systèmes de commande échantillonnent les mesures sur les processus et émettent leurs commandes simultanément. Nous allons calculer à chaque instant les lois de commande optimale en résolvant les problèmes locaux. La coordination sera effectuée en utilisant les trajectoires d'état et d'état adjoint obtenues lors des calculs effectués à l'instant précédent.

Plaçons-nous à l'instant m au niveau du processus P_j (figure 2) :

Nous venons de calculer pour les processus P_j les commandes $\hat{u}_{j,m-1}^o$ que nous appliquons à l'instant m :

$$\underline{u}_j^m = \hat{u}_{j,m-1}^o .$$

Nous disposons également des trajectoires d'état et d'état adjoint correspondantes :

$$\{x_{j,m-1}^o, \dots, x_{j,m-1}^N\}$$

$$\{\lambda_{j,m-1}^N, \dots, \lambda_{j,m-1}^o\}$$

Si l'horizon est suffisamment grand ces trajectoires sont une assez bonne prédiction des trajectoires qui seront calculées pendant la période $m, m+1$ pour l'horizon $m+1, m+N$:

$$\{x_{j,m}^o, \dots, x_{j,m}^{N-1}\} \simeq \{x_{j,m-1}^1, \dots, x_{j,m-1}^N\}$$

$$\{\lambda_{j,m}^{N-1}, \dots, \lambda_{j,m}^o\} \simeq \{\lambda_{j,m-1}^N, \dots, \lambda_{j,m-1}^1\}$$

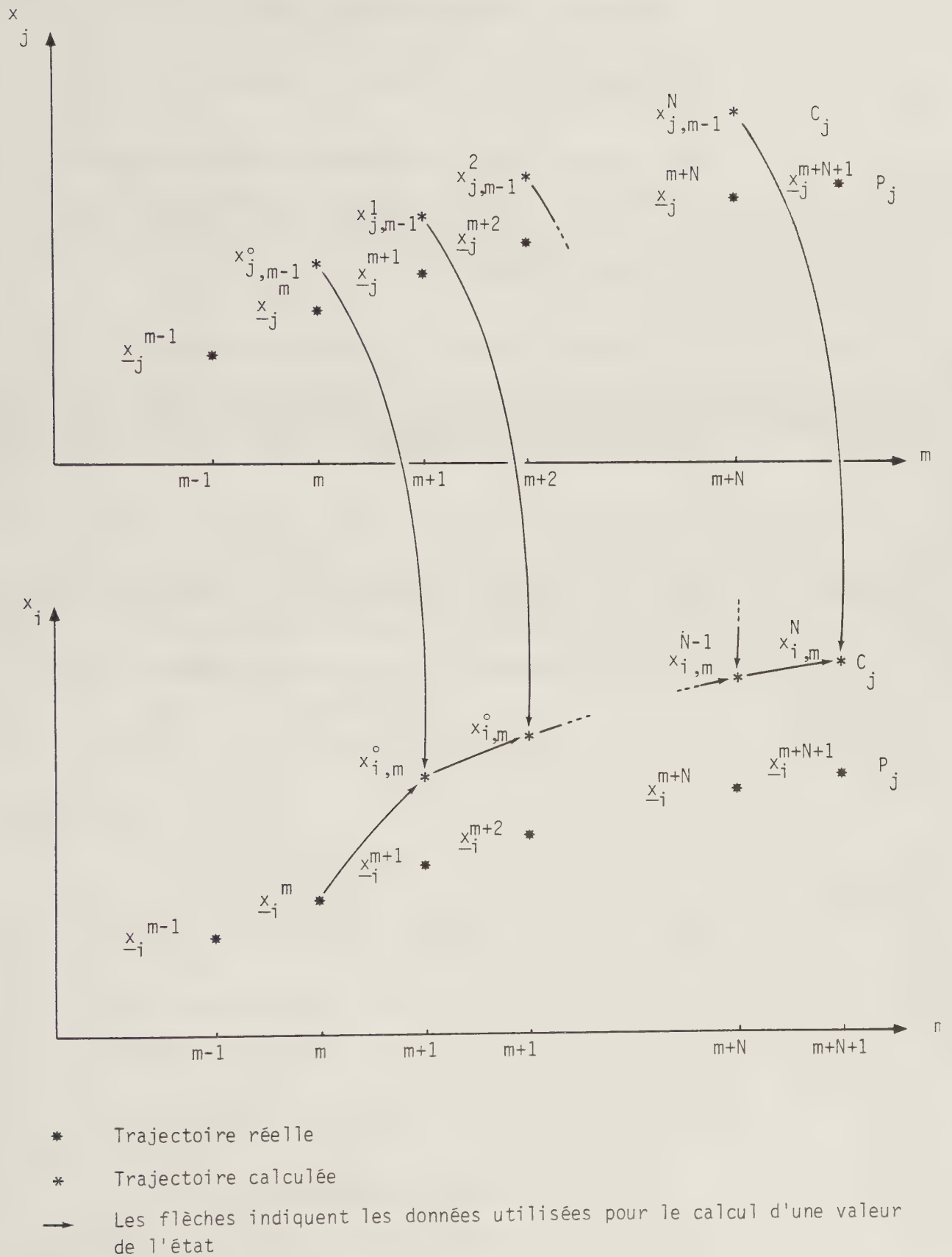


Figure 2 : Coordination des calculs (état)

Nous voulons calculer la commande à appliquer au processus P_i entre les instants $m+1$ et $m+2$:

$$\underline{u}_i^{m+1} = \hat{u}_{i,m}^o$$

Nous prédisons l'état à l'instant $m+1$, $\underline{x}_i^{m+1}$ à partir de la mesure de l'état réel $\underline{x}_i^m$, de la commande $\underline{u}_i^m$ ($\hat{u}_{i,m-1}^o$) et des valeurs $\underline{x}_{j,m-1}^o$ (prédiction de $\underline{x}_j^m$) calculées au cours de la période précédente :

$$(3.2) \quad \underline{x}_i^{m+1} = A_{ii} \underline{x}_i^m + B_{ii} \underline{u}_i^m + \sum_{j \neq i} A_{ij} \underline{x}_{j,m-1}^o$$

Nous déterminons la loi de commande optimale $\{\hat{u}_{i,m}^n\}$ en résolvant le système d'équations :

$$(3.3) \quad \begin{cases} \underline{x}_{i,m}^{n+1} = A_{ii} \underline{x}_{i,m}^n + B_{ii} \underline{u}_{i,m}^n + e_{i,m}^n & \underline{x}_{i,m}^o = \underline{x}_i^{m+1} & n=0, \dots, N-1 \\ \lambda_{i,m}^{n-1} = P_{ii} \underline{x}_{i,m}^n + A_{ii}^T \lambda_{i,m}^n + f_{i,m}^n & \lambda_{i,m}^N = 0 & n=N, \dots, 1 \end{cases}$$

la loi de commande $\{\underline{u}_{i,m}^n\}$ étant obtenue comme en (2.10).

Les trajectoires de couplage :

$$\{e_{i,m}^o, \dots, e_{i,m}^{N-1}\}, \{f_{i,m}^N, \dots, f_{i,m}^1\}$$

sont calculées à partir de :

$$\{\underline{x}_{j,m-1}^1, \dots, \underline{x}_{j,m-1}^N\}, \{0, \lambda_{j,m-1}^N, \dots, \lambda_{j,m-1}^2\}$$

suivant les relations :

$$e_{i,m}^n = \sum_{j \neq i} A_{ij} \underline{x}_{j,m-1}^{n+1}, \quad f_{i,m}^n = \sum_{j \neq i} A_{ji}^T \lambda_{j,m-1}^{n+1}$$

Si la réitération de cette procédure ne diverge pas c'est-à-dire si à chaque instant $m, m+1, \dots, \infty$ la prédiction effectuée s'avère bonne le processus global est conduit de façon quasi optimale en boucle fermée par intermittence mais les calculs nécessaires correspondent à chaque instant à un problème local.

Cette méthode nécessite la synchronisation des systèmes de commande et l'échange d'informations à chaque instant m , ce qui peut présenter des inconvénients en cas de panne du réseau de transmission ou d'un système.

3.2.2. Fonctionnement asynchrone

Les systèmes de commande fonctionnent dans cette configuration de manière totalement asynchrone et utilisent chacun leur propre horloge. Chaque fois qu'un système émet une commande vers son processus il envoie également vers les autres calculateurs les trajectoires d'état et d'état adjoint utilisées pour la coordination. A partir de l'état acquis sur le processus et des nouvelles trajectoires de couplage communiquées par les autres systèmes de commande un nouveau calcul local est alors entrepris.

Le système peut également admettre un mode de fonctionnement dégradé lorsque le réseau de transmission n'assure pas sa fonction de façon satisfaisante. Les échanges sont alors partiellement ou totalement interrompus et chaque calculateur utilise les dernières trajectoires de couplage qui lui ont été communiquées. Pour tenir compte en partie du retard introduit les trajectoires sont décalées progressivement pour ne conserver, si l'anomalie persiste, que les valeurs finales.

3.3. Méthode mixte

Les méthodes sous-optimales ne permettent peut-être pas toujours une conduite satisfaisante du processus. Il est donc nécessaire d'utiliser, lorsque cela est possible, la méthode de commande optimale.

Cette méthode n'est cependant pas toujours applicable en permanence pour des raisons diverses, par exemple :

- la convergence du calcul devient trop lente ;
- le système de communication présente une défaillance ;
- un système de commande fonctionne anormalement.

Il est alors indispensable de choisir une nouvelle méthode de conduite, sous-optimale, assurant un fonctionnement admissible du processus.

Nous pouvons donc définir une méthode de commande en associant à une méthode optimale une (ou plusieurs) méthode(s) sous-optimale(s) destinée(s) à remplacer la méthode initiale en cas d'incident.

CONCLUSION

Nous venons de présenter une méthode de décomposition du calcul de la loi de commande optimale pour des systèmes interconnectés caractérisée par la possibilité d'un fonctionnement asynchrone des calculateurs et par la simplicité des calculs nécessaires à la coordination.

Nous en avons tiré des méthodes de conduite décentralisée d'un processus avec coordination horizontale par un système de commande réparti sur le site industriel.

Nous allons maintenant passer en revue les systèmes informatiques permettant la commande en ligne de processus complexes et rapides. Nous choisirons une architecture capable d'effectuer, en un temps très bref, les calculs nécessaires puis nous la placerons dans un système de conduite optimale décentralisé.

III - SYSTEMES INFORMATIQUES POUR LA COMMANDE OPTIMALE

1 - RAPPELS

1.1. Calcul parallèle

1.1.1. Les calculateurs multiprocesseurs

1.1.2. Les calculateurs scientifiques matriciels

1.2. Systèmes distribués

2 - CALCULATEUR PARALLELE POUR LE CALCUL DE LA LOI DE COMMANDE OPTIMALE

2.1. Calculateur multiprocesseur

2.2. Calculateur matriciel

2.3. Calculateur vecteur spécialisé

2.4. Système multiprocesseur vecteur

3 - SYSTEME DECENTRALISE DE COMMANDE OPTIMALE

3.1. Architecture du système

3.2. Calcul de la loi de commande

III.1

Nous allons nous intéresser dans ce chapitre aux moyens informatiques permettant l'application des méthodes décrites précédemment à la conduite optimale de grands systèmes.

Nous dressons d'abord un tableau des architectures informatiques destinées aux calculs rapides ou aux traitements décentralisés.

Nous choisissons ensuite le type de calculateur le plus apte au calcul en ligne de la loi de commande optimale.

Enfin nous décrivons brièvement un système de conduite optimale décentralisé pour la commande de processus interconnectés.

1 - RAPPELS

Nous allons traiter dans ce paragraphe, d'une part des calculateurs possédant des éléments de traitement fonctionnant en parallèle, d'autre part de la distribution géographique des systèmes informatiques.

1.1. Calcul parallèle

La nécessité de réaliser des calculateurs de plus en plus puissants a conduit à la conception d'architectures permettant d'effectuer plusieurs traitements en parallèle.

Les différents types d'architecture ont fait l'objet de multiples classifications dont la plus connue est celle de Flynn, basée sur le nombre d'instructions et de données traitées simultanément [10]:

- SISD (Single instruction, single data stream). L'architecture d'un calculateur monoprocesseur exécutant, à chaque instant une seule instruction sur une seule donnée
- MISD (Multiple instruction, single data stream). Plusieurs instructions sont exécutées sur une même information ; une architecture du type pipe-line [19] est rattachée à ce type par certains auteurs

- SIMD (Single instruction, multiple data stream). Une seule instruction commande le traitement simultané de plusieurs données. Les ordinateurs les plus puissants sont généralement de ce type (exemple : ILLIAC IV [22])
- MIMD (Multiple instruction, multiple data stream). Cette catégorie est celle des systèmes multiprocesseurs. Chaque processeur effectue, indépendamment des autres, les calculs qui lui ont été assignés sur les données correspondantes.

Nous allons nous intéresser plus particulièrement aux calculateurs les plus couramment employés, d'une part les systèmes multiprocesseurs, d'autre part les calculateurs spécialisés pour les calculs scientifiques rapides.

1.1.1.1. *Les calculateurs multiprocesseurs*

Nous entendons par processeur (ou processeur logique) l'ensemble des éléments permettant de réaliser un traitement de données (par exemple : processeur physique, mémoire de programme, mémoire de données propres). Un ordinateur multiprocesseur est formé par la réunion de plusieurs processeurs en vue d'effectuer un traitement global, sous le contrôle d'un système d'exploitation unique.

Le transfert d'informations entre les différents processeurs est réalisé le plus fréquemment en déposant les données à échanger dans une mémoire commune. L'accès à cette mémoire ainsi qu'au système d'entrée-sortie général s'effectue de différentes façons :

- a) L'utilisation d'un bus commun est la plus simple à mettre en oeuvre et permet une extension aisée du ordinateur. Le bus ne pouvant être utilisé, à un instant donné, que par un seul processeur, le débit d'informations est limité. Cette architecture est donc réservée aux systèmes ne comportant qu'un nombre limité de processeurs ou demandant peu d'échanges.

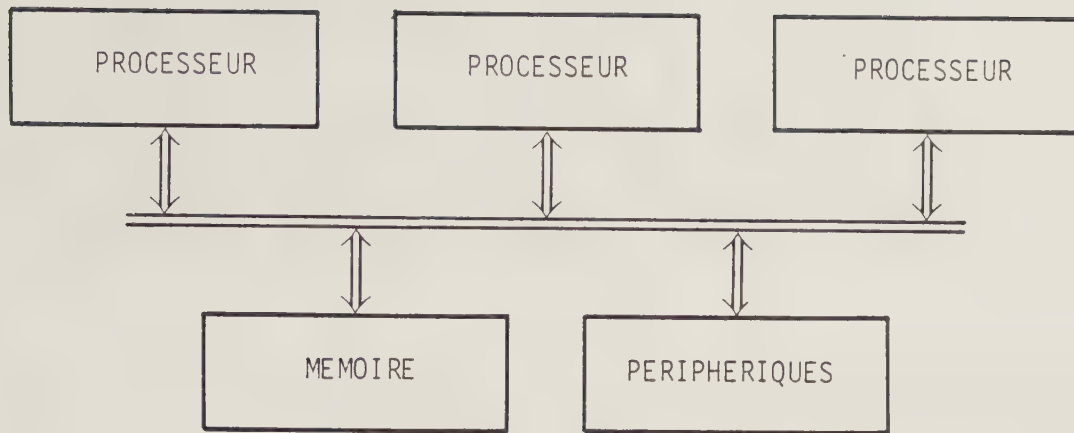


Figure 1 : Bus commun

- b) L'accès à la mémoire commune peut également être réalisé à travers une interface mettant en communication la mémoire avec les processeurs. Les performances de cette structure sont limitées par le temps d'accès à la mémoire (utilisation d'une mémoire rapide) mais sont supérieures à celle du bus commun.

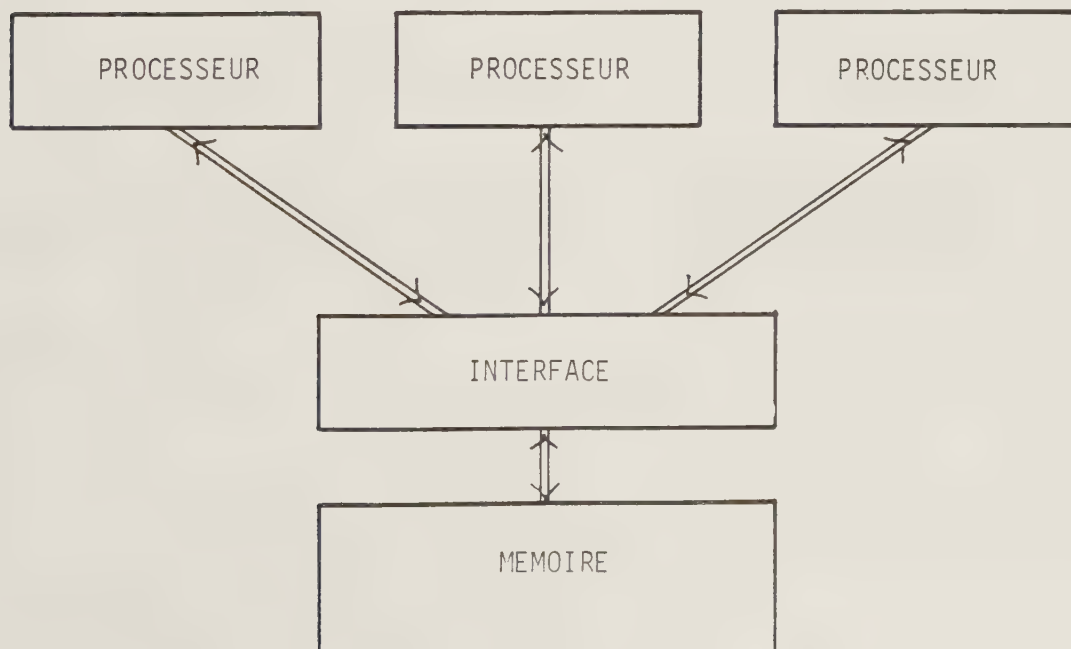


Figure 2 : Interface

- c) La solution la plus souple et la plus performante consiste à décomposer la mémoire en plusieurs blocs réunis aux processeurs par une matrice de commutation (cross-bar). Plusieurs processeurs peuvent accéder simultanément à la mémoire (chaque processeur étant connecté à un bloc) ce qui autorise des échanges très rapides. La complexité de la matrice de commutation croît avec le nombre d'éléments à connecter et rend ce principe coûteux.

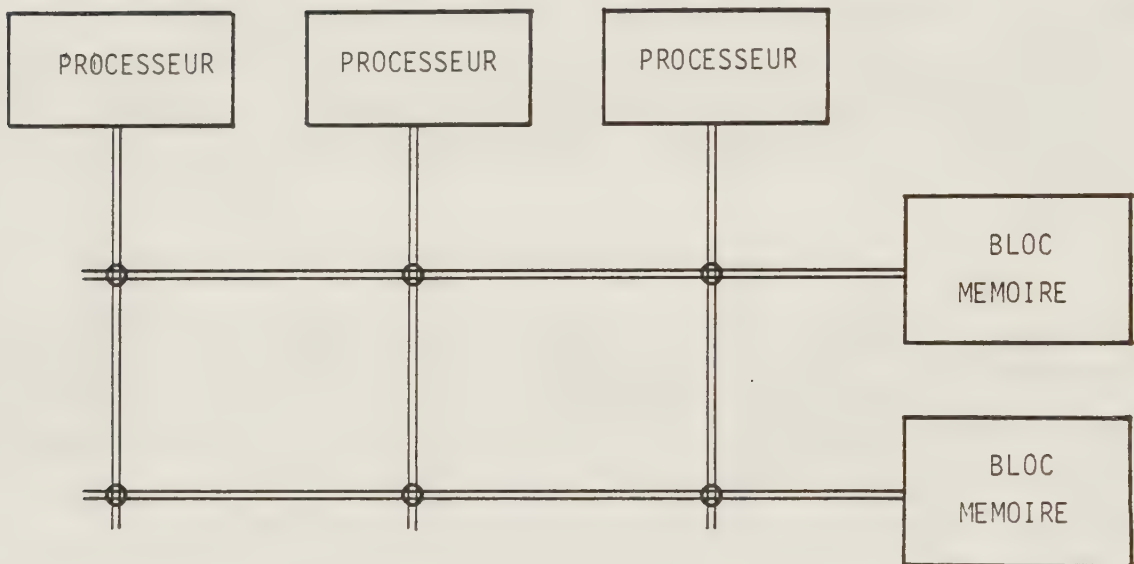


Figure 3 : Matrice de commutation

Un système de contrôle de l'accès à la mémoire (ou au bus) est nécessaire pour éviter les conflits entre les différents processeurs.

Avant de traiter un problème sur un ordinateur multiprocesseur il faut le décomposer en tâches, pouvant être exécutées en parallèle, qui seront réparties sur les processeurs.

1.1.2. Les calculateurs scientifiques matriciels

Le traitement rapide de problèmes scientifiques habituellement résolus sur des ordinateurs très puissants peut maintenant être effectué sur un ordinateur spécialisé appelé "array processor".

Associé à un miniordinateur hôte contrôlant les échanges d'informations et dirigeant le travail, ce type de calculateur réalise les calculs complexes et répétitifs [23] .

Afin de conserver une précision suffisante au cours de calculs importants les nombres sont représentés en virgule flottante par des mots de 32 ou même 64 bits. La rapidité du calculateur est obtenue par différents moyens :

- Utilisation d'opérateurs câblés performants pouvant travailler en parallèle (par exemple : multiplieur et additionneur flottants)
- Architecture du type pipe-line : l'exécution d'une nouvelle instruction peut être commencée alors que la précédente n'est pas terminée (cette technique permet par exemple de multiplier apparemment par trois les performances d'un multiplieur à trois étages)
- Logiciel permettant d'exploiter au maximum cette architecture en prévoyant le chevauchement et le lancement simultané d'instructions.
- Architecture du type MIMD réunissant plusieurs processeurs fonctionnant suivant un mode asynchrone.

Différentes possibilités sont offertes pour la programmation du calculateur afin de résoudre un problème donné :

- Programmation dans le langage assembleur du calculateur matriciel : cette méthode est la plus performante car elle permet d'utiliser au maximum les possibilités de l'architecture.
- Recours aux sous-programmes fournis par le constructeur pour les problèmes les plus courants (calcul matriciel, F F T ...) : le fonctionnement du calculateur matriciel est alors totalement transparent pour l'utilisateur qui se limite à appeler depuis son programme en langage évolué, exécuté sur le calculateur hôte, les routines de calcul existantes nécessaires à son application.

- Compilation d'un programme en langage évolué produisant le code objet exploitable par le calculateur matriciel : la facilité de programmation est privilégiée aux dépens de la vitesse d'exécution.

Ce type de calculateur est particulièrement bien adapté aux calculs itératifs et matriciels. Les performances obtenues le rendent apte aux traitements complexes en ligne.

1.2. Systèmes distribués

Pour rendre plus souple un système informatique complexe il est intéressant de répartir ses éléments sur les différents lieux d'utilisation. Nous considérons des systèmes composés de calculateurs, munis de leur propre système d'exploitation, coopérant à l'obtention d'un résultat global. Chaque ordinateur est muni de moyens de communication avec l'extérieur et avec les autres calculateurs. Les échanges d'informations sont peu nombreux et sont réalisés le plus souvent par l'une des deux méthodes suivantes :

- La connexion en anneau : chaque calculateur reçoit des informations du calculateur précédent et en transmet au suivant (figure 4). Chaque interface (I) est chargée de réémettre la donnée reçue pour la faire circuler sur l'anneau ou d'introduire une nouvelle donnée en provenance de son calculateur. L'interface assure également la réception des informations destinées à l'ordinateur associé.

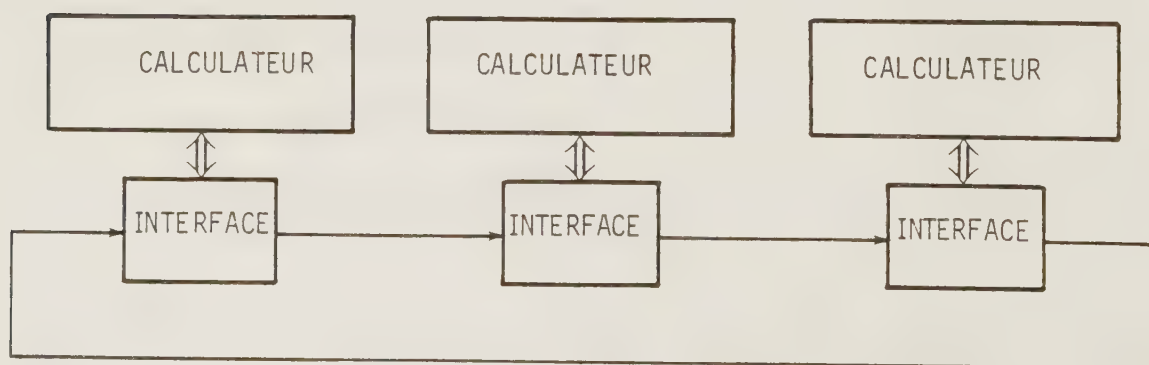


Figure 4 : Connexion en anneau

- La connexion par une ligne commune : chaque calculateur peut émettre directement une information vers tous les autres. A un instant donné une seule interface (I) envoie un message sur la ligne les autres étant alors en position d'attente afin de déterminer si le message est destiné à leur calculateur.

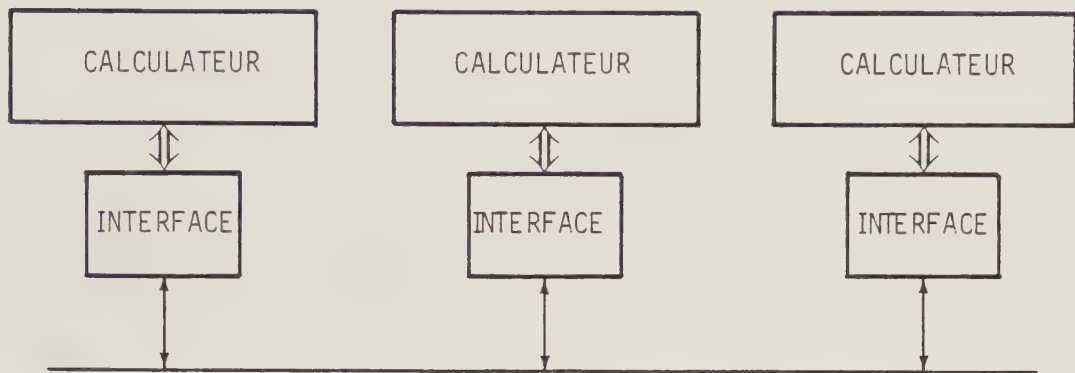


Figure 5 : Connexion par ligne commune

Les échanges sont gérés suivant un protocole de communication permettant de résoudre les conflits d'accès au système de connexion des calculateurs.

2 - CALCULATEUR PARALLELE POUR LE CALCUL DE LA LOI DE COMMANDE OPTIMALE

Nous recherchons un calculateur capable d'effectuer en un temps suffisamment bref les traitements nécessaires au calcul de la loi de commande optimale. Le parallélisme apparaît explicitement à deux niveaux :

- Le calcul porte sur des vecteurs dont les différentes composantes peuvent être traitées simultanément.
- Le calcul se décompose éventuellement en sous-problèmes résolus en parallèle.

Nous allons donc examiner les possibilités que présente l'utilisation des architectures informatiques décrites précédemment.

2.1. Calculateur multiprocesseur

L'architecture multiprocesseur convient bien à la résolution d'un problème décomposable en sous-problèmes peu couplés. Chaque sous-problème de la forme :

$$(2-1) \quad \left\{ \begin{array}{ll} x_i^{n+1} = A_{ii} x_i^n + B_{ii} u_i^n + e_i^n & x_i^0 \text{ fixé}, \quad n = 0, \dots, N-1 \\ \lambda_i^{n-1} = P_{ii} x_i^n + A_{ii}^T \lambda_i^n + f_i^n & \lambda_i^N = 0, \quad n = N, \dots, 1 \\ v_i^n = -Q_{ii}^{-1} B_{ii}^T \lambda_i^n & \\ \\ u_{ik}^n = v_{ik}^n & \text{si } u_{ik} \min \leq v_{ik}^n \leq u_{ik} \max \\ u_{ik}^n = u_{ik} \max & \text{si } v_{ik}^n > u_{ik} \max \\ u_{ik}^n = u_{ik} \min & \text{si } v_{ik}^n < u_{ik} \min \end{array} \right.$$

est résolu par un processeur. A la fin du calcul local ce processeur place dans la mémoire commune les trajectoires $\{x_i^n\}$ et $\{\lambda_i^n\}$ qu'il vient de calculer et va prendre en échange les trajectoires $\{x_j^n\}$ et $\{\lambda_j^n\}$ ($j \neq i$) servant au calcul des trajectoires de couplage $\{e_i^n\}$ et $\{f_i^n\}$:

$$(2-2) \quad \left\{ \begin{array}{l} e_i^n = \sum_{j \neq i} A_{ij} x_j^n \\ f_i^n = \sum_{j \neq i} A_{ji}^T \lambda_j^n \end{array} \right.$$

Afin de limiter les échanges seules les composantes des x_i^n et λ_i^n réellement utiles aux autres processeurs sont rangées dans la mémoire ; de même seules les composantes des x_j^n et λ_j^n intervenant effectivement dans le calcul des termes de couplage sont chargées depuis la mémoire.

L'algorithme de relaxation chaotique à retards permet un fonctionnement totalement asynchrone des processeurs et une grande souplesse dans la gestion des échanges.

La réalisation matérielle du calculateur à partir de cartes microprocesseurs, connectées à la mémoire commune par un bus, serait relativement peu coûteuse. Malheureusement la faible puissance de calcul des microprocesseurs actuellement disponibles ne permet pas la commande de processus rapides.

2.2. Calculateur "matriciel"

Le calculateur scientifique matriciel est parfaitement adapté à la résolution d'un problème de commande optimale en un temps très bref. La programmation en langage assembleur de calculs vectoriels itératifs conduit en effet à des performances tout à fait exceptionnelles pour les calculateurs les plus puissants. Le principal obstacle à son utilisation est certainement son coût dû en partie à sa polyvalence et sa précision de calcul.

2.3. Calculateur vecteur spécialisé

Nous nous sommes donc orientés vers une architecture spécialisée du type SIMD. Nous avons conçu un calculateur comprenant un seul processeur-vecteur (ne manipulant pas des nombres mais des vecteurs) destiné au calcul global de la loi de commande d'un sous-système, ou du système complet (selon la dimension du vecteur). Nous obtenons ainsi une grande puissance de calcul pour un coût modéré grâce à la spécialisation et à la simplicité de l'architecture. Ce calculateur, décrit dans le quatrième chapitre, sera utilisé à l'intérieur d'un système de calcul ou de commande "hôte".

2.4. Système multi "processeur-vecteur"

Le but de cette architecture est de réunir les avantages du calculateur multiprocesseur et du calculateur vecteur. Le problème initial est décomposé en sous-problèmes, chacun étant traité sur un processeur vecteur. Pour des problèmes peu couplés (les matrices de couplage sont largement vides) les échanges sont peu nombreux et une architecture multiprocesseur classique est satisfaisante. Dans le cas contraire les transferts d'informations ralentissent beaucoup le fonctionnement des processeurs-vecteur.

Afin de remédier à cet inconvénient un processeur de remise à jour actualise en permanence les termes de couplage et évite ainsi l'interruption prolongée des calculs locaux (figure 6).

Chaque fois que la solution d'un problème local i a été obtenue par le processeur-vecteur associé C_i ce dernier communique les variations $\{\Delta x_i^n\}$ et $\{\Delta \lambda_i^n\}$ des trajectoires $\{x_i^n\}$ et $\{\lambda_i^n\}$ entre les deux derniers calculs. Le processeur de remise à jour détermine alors les variations correspondantes des termes de couplage :

$$(2-3) \quad \begin{cases} \Delta_i e_j^n = A_{ji} \Delta x_i^n \\ \Delta_i f_j^n = A_{ij}^T \Delta \lambda_i^n \end{cases}$$

Pour ne pas ralentir le travail du processeur P_j les termes de couplage e_j^n et f_j^n ne sont pas alors modifiés, le processeur de remise à jour agissant sur des images $e_j'^n$ et $f_j'^n$:

$$(2-4) \quad \begin{cases} (e_j'^n)^+ = e_j'^n + \Delta_i e_j^n \\ (f_j'^n)^+ = f_j'^n + \Delta_i f_j^n \end{cases}$$

Lorsque le processeur C_j aura résolu à son tour son problème local les valeurs à jour $e_j'^n$ et $f_j'^n$ viendront remplacer les anciennes valeurs e_j^n et f_j^n .

En vue d'obtenir une rapidité cohérente avec les performances des processeurs-vecteur le processeur de remise à jour possède également une architecture de type SIMD.

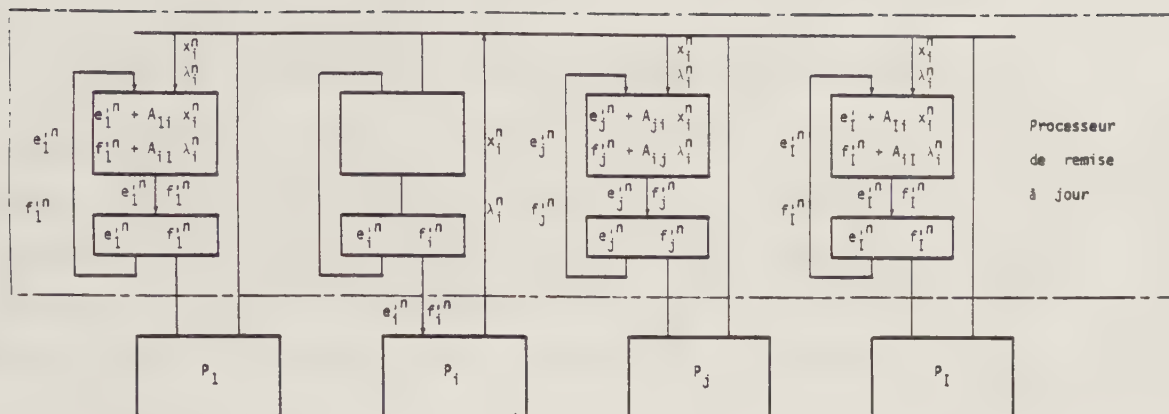


Figure 6 : Echanges après la résolution du problème local i

L'architecture multi "processeur-vecteur" ne possède pas une puissance de calcul intrinsèque plus grande qu'un calculateur vecteur de dimension égale (même nombre d'unités de calcul) alors qu'elle est plus complexe à réaliser. La présence du processeur de remise à jour n'améliore les performances du système qu'au prix d'une taille accrue. Son intérêt se limite en principe à la résolution d'un problème pouvant se décomposer en sous-problèmes peu couplés lorsque la décomposition permet d'accélérer la convergence de l'algorithme [5]. Cependant en pratique des impératifs technologiques limitent la taille du calculateur vecteur et conduisent éventuellement à scinder celui-ci en éléments plus modestes, donc à réaliser un calculateur multi "processeur-vecteur".

Nous pensons que cette architecture, théoriquement intéressante, n'est pas d'un usage suffisamment général pour que sa réalisation pratique soit envisagée dans l'immédiat.

3 - SYSTEME DECENTRALISE DE COMMANDE OPTIMALE

Un calculateur unique, si puissant soit-il, est mal adapté à la conduite d'un ensemble de processus interconnectés répartis sur un site industriel pour plusieurs raisons :

- un défaut de fonctionnement affecte l'ensemble de l'installation
- l'implantation sur le site est complexe
- les améliorations éventuelles sont difficiles à réaliser

L'exploitation de la structure naturelle de l'installation élimine ces inconvénients tout en permettant une meilleure appréhension du problème. Nous allons donc décrire rapidement un système informatique distribué pour la commande optimale puis nous exposerons la mise en oeuvre de la méthode de calcul avancée au chapitre précédent.

3.1. Architecture du système (figure 7)

Chaque processus P_i est conduit par un calculateur C_i (microordinateur) assurant les fonctions suivantes :

- Acquisition de l'état
- Emission de la commande
- Traitement des alarmes

Le calcul de la commande est effectué par un calculateur-vecteur CV_i beaucoup plus performant pour cette fonction. L'ensemble peut éventuellement commander le processus P_i de façon tout à fait autonome.

Les processus P_i étant interconnectés les calculateurs C_i seront en outre chargés d'assurer la coordination nécessaire à une conduite optimale du processus global. Aux fonctions citées plus haut s'ajoutent donc :

- Emission des trajectoires $\{x_i^n\}$ et $\{\lambda_i^n\}$ vers les autres calculateurs
- Calcul des termes de couplage e_i^n et f_i^n .

Un calculateur central est chargé de rassembler les différentes informations concernant le processus et de superviser l'activité du système.

La connexion des calculateurs est réalisée par une ligne commune plus fiable du fait de son caractère passif (une interface défectueuse ne paralyse pas le système). Les échanges sont effectués sous le contrôle du calculateur central apte à prendre les décisions importantes grâce à sa faculté d'analyser le comportement global des processus. Afin de limiter le nombre de liaisons l'affectation de la ligne est assurée par un message, envoyé par le calculateur central, autorisant un calculateur local à émettre ; celui-ci répond puis redonne le contrôle de la ligne au calculateur central.

La sécurité de fonctionnement est assurée par différents moyens :

- Chaque interface I_i vérifie si les données qu'elle émet sont présentes sur la ligne et interrompt son émission dans le cas contraire.
- Lorsque le calculateur central constate une anomalie il bloque la ligne qui est alors libérée par les calculateurs locaux.
- Enfin quand un calculateur est isolé pendant un temps trop long il adopte un mode de fonctionnement autonome.

Cette architecture est destinée à la conduite optimale globale d'un ensemble de processus interconnectés. En cas de défaillance du système de transmission un mode de fonctionnement dégradé (sous-optimal) est mis en oeuvre. Lors d'une interruption complète des communications chaque calculateur local assure la commande (optimale locale) du processus qui lui est associé.

3.2. Calcul de la loi de commande

Nous allons examiner comment s'effectue sur ce système le calcul de la loi de commande optimale.

Le problème est ici naturellement décomposé en sous-problèmes et nous allons adopter la méthode de calcul décrite pour le calculateur multiprocesseur (paragraphe 2.1).

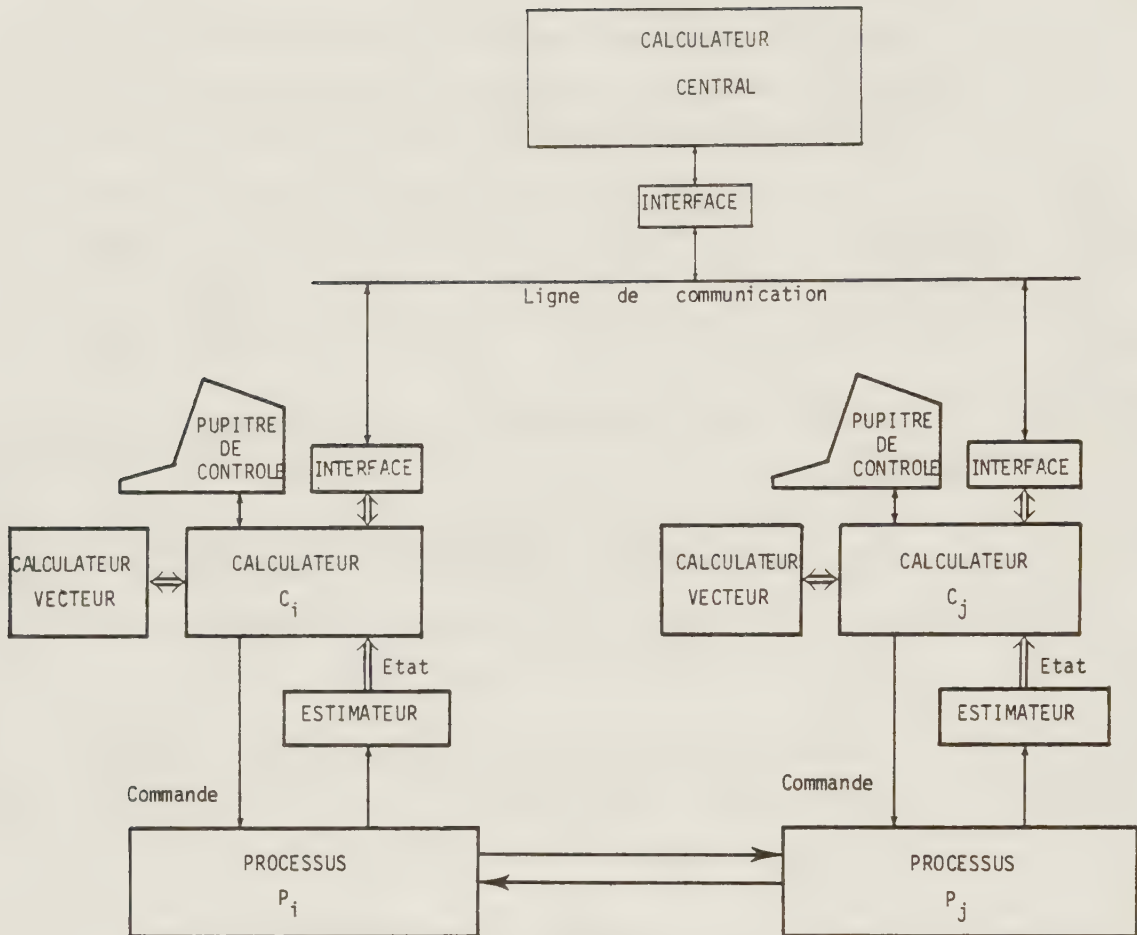


Figure 7 : Conduite décentralisée de processus interconnectés

Le $i^{\text{ème}}$ sous problème est résolu par le calculateur vecteur associé CV_i alors que les termes de couplage sont fournis par le calculateur C_i .

Les échanges s'effectuent ici d'une façon différente puisqu'il n'existe pas de mémoire commune. Lorsque la solution du $i^{\text{ème}}$ sous-problème est obtenue le calculateur C_i envoie vers les autres calculateurs les trajectoires $\{x_i^n\}$ et $\{\lambda_i^n\}$ qui seront rangées dans leur propre mémoire. Il calcule ensuite les termes de couplage e_j^n et f_j^n , en utilisant les valeurs x_j^n et λ_j^n ($j \neq i$) reçues précédemment, et les communique à son calculateur vecteur afin de lancer un nouveau calcul. Le calculateur central examine les différents échanges afin de détecter la convergence

du calcul global pour l'arrêter. Il commande également périodiquement l'émission de la commande et l'acquisition de l'état puis le lancement d'un nouveau calcul.

L'algorithme de relaxation chaotique à retard utilisé n'est pas déterminé à l'avance. Il dépend du temps nécessaire à la résolution d'un sous-problème ainsi que des priorités affectées aux différents calculateurs pour les échanges.

Les calculateurs C_j sont peu sollicités pour le calcul de la loi de commande optimale car les processus sont supposés peu couplés. Pour alléger le travail de chaque calculateur seules les trajectoires utiles sont communiquées aux autres calculateurs. D'autre part l'interface de communication assure la sélection des valeurs nécessaires au calcul des termes de couplage, parmi les données présentes sur la ligne, libérant ainsi le calculateur pour d'autres tâches.

CONCLUSION

Nous nous orientons donc vers la réalisation d'un calculateur vecteur spécialisé pour le calcul de la loi de commande optimale. Ce calculateur est conçu de façon à pouvoir prendre place dans un système de commande optimale décentralisé adapté à la conduite de processus interconnectés.

IV - CARACTERISTIQUES DU CALCULATEUR VECTEUR

1 - PROBLEME A TRAITER

2 - CALCULATEUR VECTEUR

3 - DIMENSION DU CALCULATEUR

4 - PRECISION

4.1. Représentation des nombres

4.2. Cadrage des variables

4.3. Erreur sur la solution

4.4. Estimation de l'erreur commise au cours d'une itération

5 - EXPERIENCES PRELIMINAIRES

Nous allons maintenant nous intéresser à la résolution d'un problème de commande optimale, en un temps réduit, par un calculateur vecteur spécialisé.

1 - PROBLEME A TRAITER

Les équations correspondant à un problème local d'optimisation dans le cas d'une résolution décentralisée sont de la forme :

$$(1-1) \left\{ \begin{array}{l} (a) \quad x^{n+1} = A x^n + B u^n + e^n, \quad x^0 = x_0, \quad n = 0, \dots, N-1 \\ (b) \quad \lambda^{n-1} = P x^n + A^T \lambda^n + f^n, \quad \lambda^n = 0, \quad n = N, \dots, 1 \\ (c) \quad \left\{ \begin{array}{ll} v^n = -Q^{-1} B^T \lambda^n & n = 0, \dots, N-1 \\ u_i^n = v_i^n & \text{si } u_{i \min} \leq v_i^n \leq u_{i \max} \\ u_i^n = u_{i \max} & \text{si } v_i^n > u_{i \max} \\ u_i^n = u_{i \min} & \text{si } v_i^n < u_{i \min} \end{array} \right. \end{array} \right.$$

e^n et f^n étant les termes de couplage.

Ces équations peuvent représenter également les équations d'optimalité d'un problème global de commande optimale.

Le calcul de la solution va être effectué en utilisant la procédure itérative suivante.

A partir d'une loi de commande estimée $\{u^n\}^0$ et de x_0 nous calculons la trajectoire $\{x^n\}^0$ en utilisant l'équation de récurrence (a). La trajectoire $\{\lambda^n\}^0$ est obtenue de façon identique à partir de $\{x^n\}^0$ et de 0 par l'équation (b). La nouvelle loi de commande $\{u^n\}^1$ résulte de l'application de (c) à $\{\lambda^n\}^0$.

Si nous nous intéressons à la résolution d'un problème global cette procédure sera réitérée jusqu'à convergence vers la solution.

Au contraire pour un problème local le nombre d'itérations à effectuer sera fixé par l'algorithme de résolution du problème global. La recherche d'une solution locale est cependant un cas particulier très intéressant (cf. II,2.2.3).

La méthode de résolution adoptée est purement séquentielle mais les différentes variables sont vectorielles. Les calculs sont accélérés en traitant en parallèle les composantes de chaque vecteur sur un calculateur vecteur.

2 - CALCULATEUR VECTEUR

Le calculateur vecteur se compose d'une unité de commande assurant la gestion d'un nombre d'unités de traitement égal à la dimension des vecteurs à manipuler. Chaque unité de traitement, associée à une composante, effectue les mémorisations et les calculs correspondants.

Les opérations vectorielles nécessaires à la résolution du problème envisagé sont obtenues en réalisant en parallèle sur les différentes unités de traitement des opérations scalaires.

La somme de deux vecteurs $X + Y$ se décompose en :

$$\begin{array}{ll} x_1 + y_1 & \text{sur l'unité de calcul 1} \\ \vdots & \\ x_I + y_I & \text{sur l'unité de calcul I} \end{array}$$

Le produit d'une matrice par un vecteur AX devient :

$$\begin{array}{ll} a_{11} x_1 + \dots + a_{1I} x_I & \text{sur l'unité de calcul 1} \\ \vdots & \\ a_{I1} x_1 + \dots + a_{II} x_I & \text{sur l'unité de calcul I} \end{array}$$

Cet exemple montre la nécessité d'un système permettant des échanges de données entre les différentes unités de traitement. Les échanges se présentent sous la forme de diffusion d'une information à partir d'une unité de traitement vers toutes les autres. Un moyen de communication avec l'extérieur sera également indispensable.

Aux opérations arithmétiques il faut ajouter deux fonctions un peu particulières : le calcul de u^n à partir de v^n et le test de convergence. La loi de commande est obtenue en conservant, après comparaison de v_i^n avec les bornes du domaine d'admissibilité $u_{i \max}$ et $u_{i \min}$, l'une de ces trois valeurs. Pour une exécution en parallèle il est nécessaire que les tests soient effectués au niveau de l'unité de traitement. Le test de convergence est effectué en comparant la norme de l'écart entre deux trajectoires $\{u^n\}$ successives à une valeur fixée. Le choix d'une norme judicieuse diminuera le temps nécessaire à cette opération.

Le programme implanté sur ce calculateur devra permettre d'une part d'effectuer les calculs, d'autre part d'assurer les échanges avec l'extérieur. En possession des coefficients des matrices et des bornes du domaine d'admissibilité le calculateur vecteur recevra l'état initial x_0 et les trajectoires de couplage $\{e^n\}$ et $\{f^n\}$, calculera et fournira la loi de commande optimale $\{\hat{u}^n\}$ solution de (1.1) ainsi que les trajectoires $\{\hat{x}^n\}$ et $\{\hat{\lambda}^n\}$.

3 - DIMENSION DU CALCULATEUR

Le calculateur vecteur sera caractérisé par son nombre d'unités de traitement et par la taille de sa mémoire. Ces deux paramètres dépendront de la dimension I du problème à résoudre et de l'horizon N .

La dimension du modèle I est le résultat d'un compromis entre l'exactitude avec laquelle le processus est représenté et le volume des calculs à effectuer pour obtenir la loi de commande optimale. Le temps de calcul, proportionnel à I^2 sur un calculateur monoprocesseur, est proportionnel à I sur le calculateur vecteur grâce à la présence I unités de traitement (les échanges ne ralentissent pas sensiblement la vitesse de calcul). Le calculateur vecteur permet donc de résoudre des

problèmes de grande dimension en un temps très bref mais en revanche des considérations de coût peuvent intervenir (chaque unité de traitement ayant la puissance de calcul d'un miniordinateur).

L'horizon N dépend de la période d'échantillonnage ΔT et de l'horizon réel T :

$$N = \frac{T}{\Delta T}$$

La période d'échantillonnage est prise assez petite pour que le modèle représente avec suffisamment d'exactitude le processus tout en permettant une conduite en boucle fermée (le calcul doit être effectué en un temps inférieur à ΔT ; cf. I.5). L'horizon réel T est choisi avec le critère lors de la détermination de l'objectif à atteindre. Il est bien sûr possible de modifier le critère et l'horizon si le choix initial entraîne des calculs trop longs.

Les dimensions du calculateur découlent donc d'un compromis entre le choix d'un problème aboutissant à une conduite performante du processus, la possibilité d'effectuer les calculs nécessaires suffisamment rapidement et le coût du calculateur.

Pour la réalisation du prototype nous avons limité la dimension à 16 et l'horizon à 31, ces valeurs permettant de vérifier le fonctionnement sans entraîner un coût trop élevé.

4 - PRECISION DU CALCULATEUR

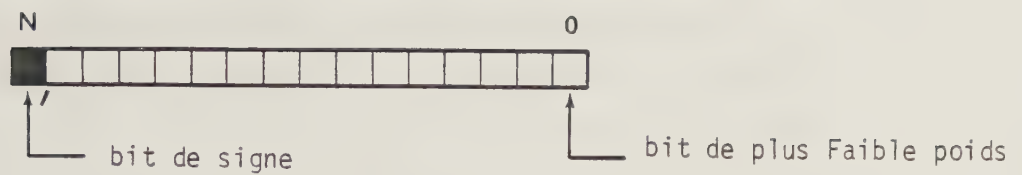
La loi de commande optimale ne sera pas obtenue avec une exactitude rigoureuse aussi nous allons estimer l'erreur commise lors de son calcul.

4.1. Représentation des nombres

Les nombres traités par le calculateur sont représentés par leur valeur binaire sous deux formes.

La plus simple est la représentation en virgule fixe :

IV.5



Les nombres doivent être cadrés entre -1 et $1 - 2^{-N}$ et la résolution est de 2^{-N} (1 bit de plus faible poids). Les erreurs de calcul sont dues aux troncatures suivant les multiplications et sont majorées par 2^{-N} .

La représentation la plus pratique est la représentation en virgule flottante :



$$x = m 2^e$$

Le nombre x est représenté par sa mantisse cadrée à gauche m et par son exposant e . Le cadrage préalable des variables n'est plus nécessaire. La mantisse restant cadrée de façon optimale la précision relative est constante et les erreurs relatives dues aux troncatures (ou aux arrondis) sont beaucoup plus faibles que lors de calculs en virgule fixe. Le calcul en virgule flottante allie donc souplesse d'emploi et précision.

Cependant la structure du calculteur vecteur ne permet pas d'effectuer facilement les opérations arithmétiques en virgule flottante. Toutes les unités de calcul effectuent la même opération

commandée par l'unité de commande alors qu'il serait nécessaire d'effectuer un traitement différent sur chaque unité de calcul. Une solution hybride consiste à représenter globalement le vecteur en virgule flottante avec un seul exposant pour toutes les composantes, le cadrage étant effectué sur la plus grande composante en valeur absolue. Mais pour de grands systèmes seul le câblage des opérations flottantes au niveau de chaque unité de calcul est une solution satisfaisante.

Pour le prototype que nous vous proposons de réaliser nous utiliserons la représentation en virgule fixe beaucoup plus simple.

4.2. Cadrage des variables

La précision obtenue lors de calculs en virgule fixe est conditionnée par le cadrage des variables et des coefficients. Les nombres doivent être cadrés le plus à gauche possible tout en restant dans l'intervalle $[-1, 1 - 2^{-N}]$.

Reprenons le système (1-1) en remarquant que A a généralement ses coefficients diagonaux voisins de 1 et ses coefficients extradiagonaux petits et en posant (annexe IV,1) :

$$A = I + \Gamma$$

$$(1-2) \quad \begin{cases} x^{n+1} = x^n + \Gamma x^n + B u^n + e^n \\ \lambda^{n+1} = \lambda^n + P x^n + \Gamma^T \lambda^n + f^n \\ v^n = -Q^{-1} B^T \lambda^n \quad \text{d'où l'on tire } u^n \end{cases}$$

Le cadrage est effectué en introduisant trois matrices diagonales α , β , γ et en utilisant les nouvelles variables X , Λ , V , U

$$X = \alpha x \quad \Lambda = \beta \lambda \quad V = \gamma v \quad U = \gamma u$$

Nous allons avoir à redoubler le système équivalent :

$$(1-3) \left\{ \begin{array}{l} X^{n+1} = X^n + \alpha \Gamma \alpha^{-1} X^n + \alpha B \gamma^{-1} U^n + \alpha e^n \quad X^0 = \alpha x^0 \\ \Lambda^n = \Lambda^n + \beta P \alpha^{-1} X^n + \beta \Gamma^T \beta^{-1} \Lambda^n + \beta f^n \quad \Lambda^N = 0 \\ V^n = -\gamma Q^{-1} B^T \beta^{-1} \Lambda^n \\ U_i^N = V_i^N \quad \text{si} \quad \gamma_i u_{i \min} < V_i^n < \gamma_i u_{i \max} \\ U_i^N = u_{i \max} \quad \text{si} \quad V_i^n > \gamma_i u_{i \max} \\ U_i^N = u_{i \min} \quad \text{si} \quad V_i^n < \gamma_i u_{i \min} \end{array} \right.$$

Les matrices α , β et γ seront choisies afin que d'une part les coefficients des matrices $\alpha \Gamma \alpha^{-1}$, $\alpha B \gamma^{-1}$, $\beta P \alpha^{-1}$, $\beta \Gamma^T \beta^{-1}$, $-\gamma Q^{-1} B^T \beta^{-1}$, d'autre part les variables X_i^n , Λ_i^n , V_i^n tout au long de leur trajectoire, restent cadrés avec une précision suffisante sans débordement.

Remarque : Dans de très rares cas le cadrage des coefficients des matrices peut être impossible. Nous prenons alors un autre cadrage en rectifiant par des décalages les résultats des calculs.

4.3. Erreur sur la solution

La solution est obtenue en recherchant le point fixe de l'application F associant à la loi de commande U_p calculée à l'itération p la loi de commande U_{p+1}

$$U_{p+1} = F(U_p)$$

La procédure itérative est poursuivie jusqu'à convergence c'est-à-dire jusqu'à ce que deux itérés successifs soient suffisamment proches. Le test effectué consiste à comparer la distance entre ces deux itérés à une valeur fixée δ :

$$||U_p - U_{p+1}|| \leq \delta$$

En fait le calcul de $F(\hat{U}_p)$ n'est pas exact et nous noterons $\hat{U}_p$ les valeurs successives obtenues par le calcul réel.

$$||\hat{U}_{p+1} - F(\hat{U}_p)|| \leq \varepsilon \quad \varepsilon \text{ est l'erreur maximale}$$

Supposons que la procédure de calcul ait "convergé" à l'itération $p+1$.

$$||\hat{U}_p - \hat{U}_{p+1}|| \leq \delta$$

L'application F est contractante, nous avons donc en notant U le point fixe de F :

$$||F(\hat{U}_p) - U|| \leq d ||\hat{U}_p - U|| \quad d \in [0,1]$$

En vertu de l'inégalité triangulaire nous obtenons :

$$||\hat{U}_p - U|| \leq \underbrace{||\hat{U}_p - \hat{U}_{p+1}||}_{\leq \delta} + \underbrace{||\hat{U}_{p+1} - F(\hat{U}_p)||}_{\leq \varepsilon} + ||F(\hat{U}_p) - U||$$

Nous tirons donc des deux dernières inéquations :

$$||F(\hat{U}_p) - U|| \leq \frac{d}{1-d} (\varepsilon + \delta)$$

En utilisant de nouveau l'inégalité triangulaire nous écrivons :

$$||\hat{U}_{p+1} - U|| \leq ||\hat{U}_{p+1} - F(\hat{U}_p)|| + ||F(\hat{U}_p) - U||$$

L'erreur commise sur la solution du problème sera donc :

$$(1-4) \quad ||\hat{U}_{p+1} - U|| \leq \frac{1}{1-d} \varepsilon + \frac{d}{1-d} \delta$$

La précision du calcul peut être majorée par une expression dépendant uniquement du test de convergence et de l'erreur obtenue au cours de la dernière itération.

4.4. Estimation de l'erreur commise au cours d'une itération

L'erreur de calcul est due à la troncature ou à l'arrondi effectué après chaque multiplication. Il est possible de conserver le résultat complet sur un mot de longueur double mais les échanges les mémorisations et les multiplications seront effectués sur des mots de simple longueur.

Partant d'une loi de commande $\{u^n\}_p$ donnée nous allons évaluer l'erreur commise au cours du calcul de $\{u^n\}_{p+1}$.

La trajectoire d'état exacte est donnée par :

$$x^{n+1} = x^n + \Gamma x^n + B u^n + e^n \quad \Gamma = A - I$$

La trajectoire effectivement calculée est $\bar{x}^n$:

$$\bar{x}^{n+1} = \bar{x}^n + \Gamma \tilde{x}^n + B u^n + e^n \quad \bar{x}^0 = \tilde{x}^0$$

$\tilde{x}^n$ représente la valeur de $\bar{x}^n$ après troncature ou arrondi :

$$\tilde{x}^n = x^n - \epsilon_x^{n-1}$$

ϵ_x^{n-1} étant l'erreur commise lors de la troncature ou de l'arrondi.

La trajectoire calculée $\bar{x}^n$ est donnée par :

$$\bar{x}^{n+1} = \bar{x}^n + \Gamma \bar{x}^n + B u^n + e^n - \Gamma \epsilon_x^{n-1}$$

L'erreur sur la trajectoire $\bar{x}^n$ est donc :

$$x^n - \bar{x}^n = \sum_{\ell=0}^{n-2} \Gamma^{n-1-\ell} \epsilon_x^\ell$$

Puis après troncature l'erreur devient :

$$E_x^n = x^n - \tilde{x}^n = \sum_{\ell=0}^{n-1} \Gamma^{n-1-\ell} \epsilon_x^\ell$$

$$\bar{\lambda}^{n-1} = \bar{\lambda}^n + P \tilde{x}^n + \Gamma^T \tilde{\lambda}^n + f^n$$

$$\bar{\lambda}^{n-1} = \bar{\lambda}^n + P x^n + \Gamma^T \bar{\lambda}^n + f^n - P E_x^n - \Gamma^T \epsilon_\lambda^{n+1}$$

$$\lambda^n - \bar{\lambda}^n = \sum_{\ell=n+1}^N \Gamma^{T(\ell-(n+1))} P E_x^\ell + \sum_{\ell=n+2}^N \Gamma^{T(\ell-(n+1))} \epsilon_\lambda^\ell$$

$$E_\lambda^n = \lambda^n - \tilde{\lambda}^n = \sum_{\ell=n+1}^N \Gamma^{T(\ell-(n+1))} \left[P E_x^\ell + \epsilon_\lambda^\ell \right]$$

En remplaçant maintenant E_x^ℓ par son expression :

$$E_\lambda^n = \sum_{\ell=n+1}^N \Gamma^{T(\ell-(n+1))} \left[P \sum_{k=0}^{\ell-1} \Gamma^{\ell-1-k} \epsilon_x^k + \epsilon_\lambda^\ell \right]$$

Il en est de même pour la trajectoire $\bar{V}^n$:

$$\bar{V}^n = -Q^{-1} B^T \tilde{\lambda}^n$$

$$V^n - \bar{V}^n = -Q^{-1} B^T E_\lambda^n$$

L'expression de l'erreur après troncature ou arrondi est finalement :

$$(1-5) \quad E_V^n = V^n - \tilde{V}^n = -Q^{-1} B^T \left\{ \sum_{\ell=n+1}^N \Gamma^{T(\ell-(n+1))} \left[P \sum_{k=0}^{\ell-1} \Gamma^{\ell-1-k} \epsilon_x^k + \epsilon_\lambda^\ell \right] \right\} + \epsilon_V^n$$

L'erreur commise sur la loi de commande u_{p+1}^n peut alors s'exprimer de la façon suivante :

$$(1-6) \quad \begin{cases} E_u^n = u_{p+1} - \tilde{u}_{p+1} \\ E_{u_i}^n = E_{V_i}^n & \text{si } u_{i \min} \leq V_i^n \leq u_{i \max} \\ |E_{u_i}^n| < |E_{V_i}^n| & \text{si } V_i^n > u_{i \max} \text{ ou } V_i^n < u_{i \min} \end{cases}$$

et par conséquent :

$$(1-7) \quad ||E_u^n|| \leq ||E_V^n||$$

Si nous supposons la période d'échantillonnage petite par rapport aux constantes de temps du processus nous utilisons le fait (annexe V) :

$$||\Gamma|| \ll 1$$

pour donner une estimation de l'erreur sur V^n :

$$(1-8) \quad E_V^n \approx -Q^{-1} B^T \left[P \varepsilon_X^n + \varepsilon_\lambda^{n+1} \right] + \varepsilon_V^n + 0$$

Notons e l'erreur de troncature ou d'arrondi maximale :

$$-e \leq \varepsilon_{X_i}^n \leq e, \quad -e \leq \varepsilon_{\lambda_i}^n \leq e, \quad -e \leq \varepsilon_{V_i}^n \leq e$$

Nous pouvons donner une estimation de l'erreur maximale obtenue dans le calcul de V^n et par conséquent de u^n :

$$(1-9) \quad ||E_V^n||_{\max} \approx e \left[\sqrt{I} ||Q^{-1} B^T P|| + \sqrt{I} ||Q^{-1} B^T|| + \sqrt{J} \right]$$

$$(1-10) \quad ||E_U^n||_{\max} \approx e \left[\sqrt{I} ||Q^{-1} B^T P|| + \sqrt{I} ||Q^{-1} B^T|| + \sqrt{J} \right]$$

En introduisant le vecteur U nous obtenons :

$$(1-11) \quad \begin{bmatrix} \tilde{U}_{p+1} - F(\tilde{U}_p) \end{bmatrix} = \begin{bmatrix} E_u^0 \\ \vdots \\ E_u^{N-1} \end{bmatrix}$$

Notons $\varepsilon_{\max}$ l'erreur maximale commise sur U en une itération :

$$(1-12) \quad \varepsilon_{\max} \approx \sqrt{N} \left[\sqrt{I} ||Q^{-1} B^T P|| + \sqrt{I} ||Q^{-1} B^T|| + \sqrt{J} \right] e$$

Si nous effectuons une troncature e est égal à la valeur représentée par le bit de plus faible poids et l'erreur commise est toujours de même signe. Si nous effectuons un arrondi e est égal à la moitié du bit de faible poids. L'avantage de l'arrondi par rapport à la troncature est dû à la moindre accumulation des erreurs.

La valeur de l'erreur $\epsilon_{\max}$ est simple à calculer mais donne une valeur très approximative et pessimiste de l'erreur réelle ϵ . Nous nous proposons donc de rechercher expérimentalement pour les différents problèmes à traiter la précision obtenue.

5 - EXPERIENCES PRELIMINAIRES

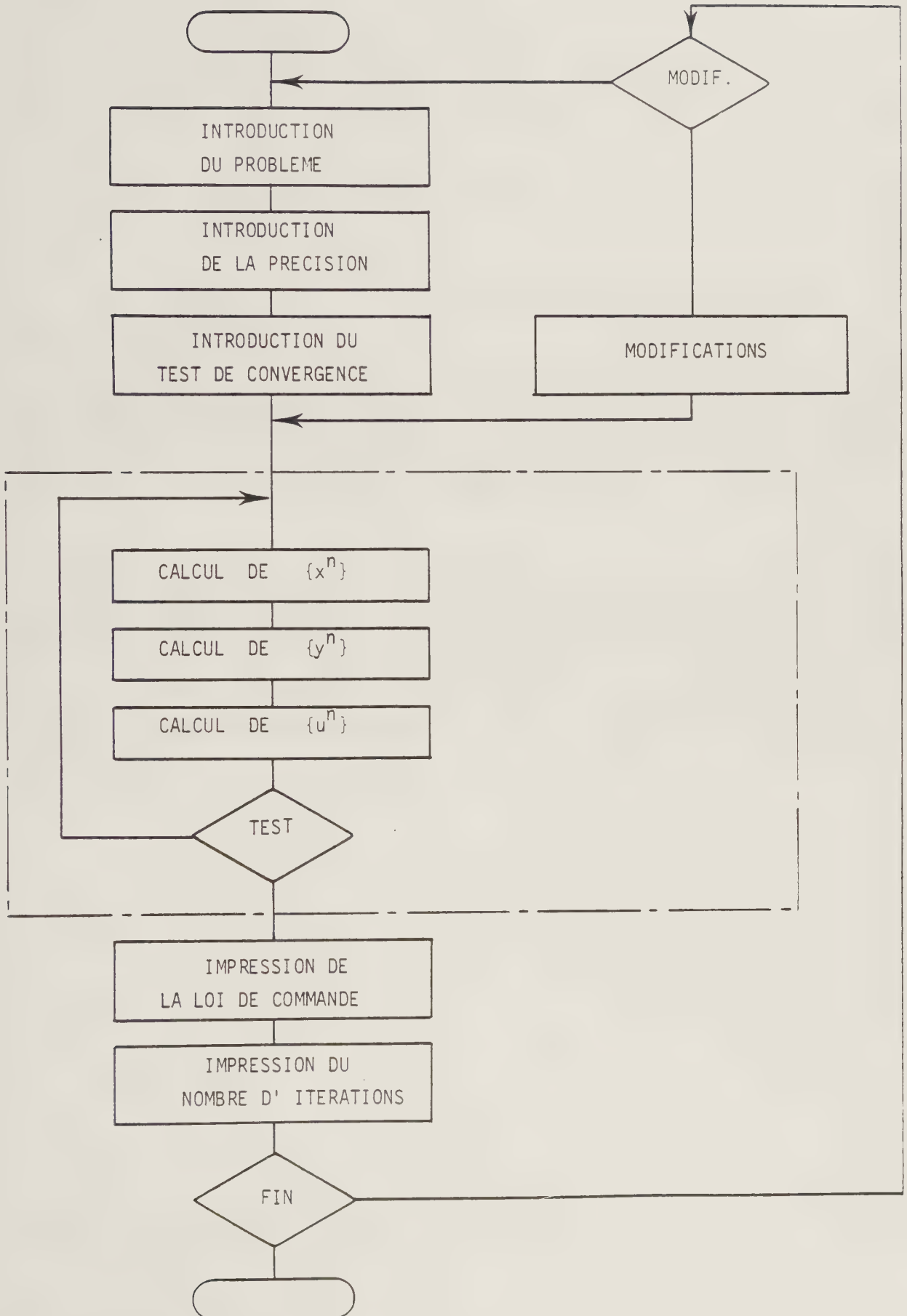
Afin d'évaluer les performances réelles du calculateur vecteur nous allons tester l'algorithme de calcul sur un système informatique classique. Nous étudierons principalement la précision obtenue et le temps nécessaire au calcul de la loi de commande optimale.

Le programme se décompose en deux parties : le programme de calcul et un programme permettant l'introduction en mode conversationnel des différents paramètres.

Les calculs sont effectués en virgule fixe, la longueur des mots étant choisie au départ par l'opérateur. Les résultats fournis sont principalement la loi de commande et le nombre d'itérations nécessaires pour l'obtenir. Pour un problème donné nous faisons varier la longueur des mots, le test de convergence ainsi que le paramètre de relaxation. Nous observons la précision obtenue par rapport au résultat exact (calculé en virgule flottante) ainsi que le nombre d'itérations nécessaires.

Le chapitre VII sera consacré à l'évaluation du temps de calcul, compte tenu des éléments technologiques utilisés, ainsi qu'à la mesure des performances réelles du système. Nous ne donnons donc ici que les conclusions obtenues à ce stade :

- La précision obtenue s'est avérée très bonne la majoration (1-8) étant bien pessimiste. L'utilisation de mots de 16 bits pour le prototype permet de traiter la plupart des problèmes.
- Le choix d'un paramètre de relaxation convenable autorise la convergence de nombreux problèmes en quelques itérations.



ANNEXE IV 1

DECOMPOSITION DE LA MATRICE A

Pour commander par un calculateur numérique un processus continu dont le modèle est :

$$\dot{y}(t) = Ay(t) + Bu(t)$$

nous utilisons le modèle récurrent correspondant :

$$y^{n+1} = e^{\Delta t A} y^n + A^{-1} (e^{\Delta t A} - 1) B u^n$$

où Δt est la période d'échantillonnage.

Pour que la représentation du processus soit suffisamment précise la période d'échantillonnage est choisie beaucoup plus faible que les constantes de temps du modèle continu. Les valeurs propres de la matrice $\Delta t A$ étant petites nous pouvons écrire :

$$A = e^{\Delta t A} \approx I + \Delta t$$

$$A = I + \Gamma \quad \Gamma \approx \Delta t A$$

avec de plus :

$$||\Gamma|| \ll 1$$

V - ARCHITECTURE DU CALCULATEUR VECTEUR

1 - GENERALITES

1.1. Calculateur microprogrammé

1.2. Calculateur vecteur

2 - L'UNITE DE COMMANDE

2.1. La mémoire et le registre pipe-line

2.2. Le séquenceur

2.3. Commande de l'adressage et des échanges

2.4. Commande des autres fonctions

3 - UNITE DE TRAITEMENT

3.1. L'unité de calcul

3.2. Rangement et transfert des données

4 - MISE EN OEUVRE

Les aspects matériel et logiciel sont, pour ce type de calculateur spécialisé, fortement liés. Ce chapitre, dans lequel nous décrivons l'architecture du système, est donc complété par le suivant, traitant de la programmation.

1 - GENERALITES

1.1. Calculateur microprogrammé [25]

Seule l'utilisation d'une architecture microprogrammée (figure 1) permet d'adapter de façon optimale le calculateur à une tâche donnée tout en simplifiant sa conception. Les différentes fonctions du calculateur sont commandées par des microinstructions rangées dans la mémoire de microprogramme. Le déroulement du microprogramme est assuré par un séquenceur au rythme de l'horloge. Pendant un cycle la microinstruction est contenue dans un registre commandant simultanément l'unité de traitement et le séquenceur. Afin de réaliser des ruptures de séquences conditionnelles le séquenceur utilise également des informations en provenance de l'unité de traitement. La durée d'une période d'horloge doit être suffisante pour permettre le cheminement des données du registre à l'unité de traitement puis au séquenceur et la sélection de la microinstruction suivante dans la mémoire (trajet critique).

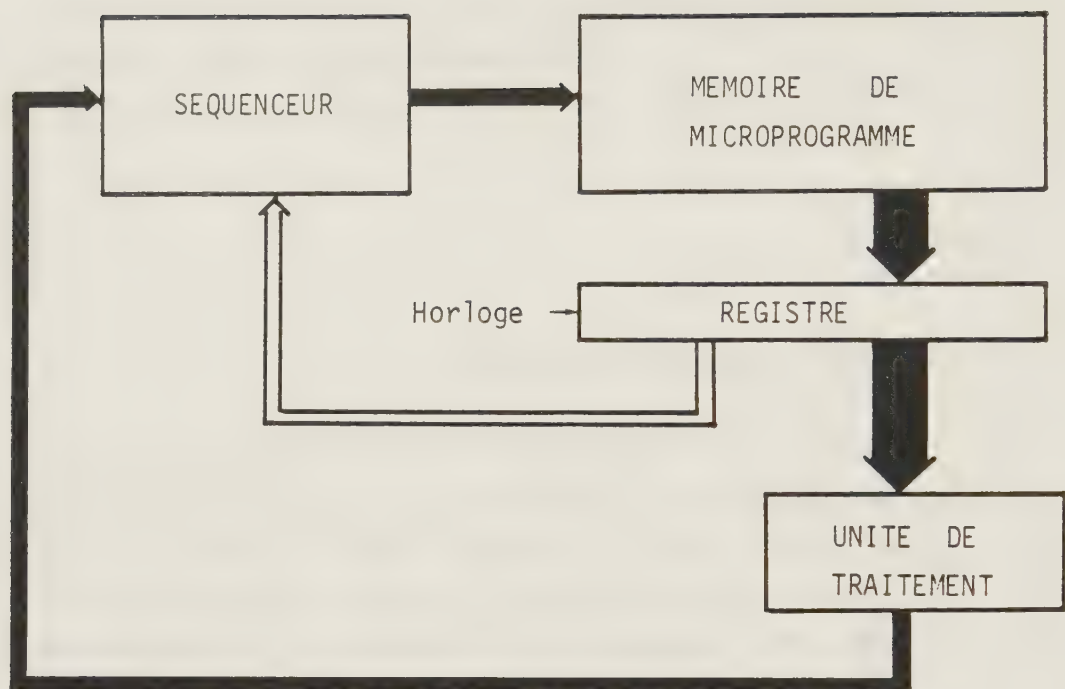


Figure 1 : Microprogrammation

La durée d'exécution des microinstructions est plus courte si l'on adopte une architecture de type pipe-line (figure 2). La mémorisation des conditions dans un registre permet au séquenceur de rechercher la microinstruction suivante pendant que l'unité de traitement effectue les calculs. La période d'horloge est déterminée par la plus longue de ces deux opérations.

La programmation est plus délicate car, au cours d'un cycle d'horloge, l'adressage de la microinstruction suivante est réalisée en fonction des résultats obtenus durant le cycle précédent.

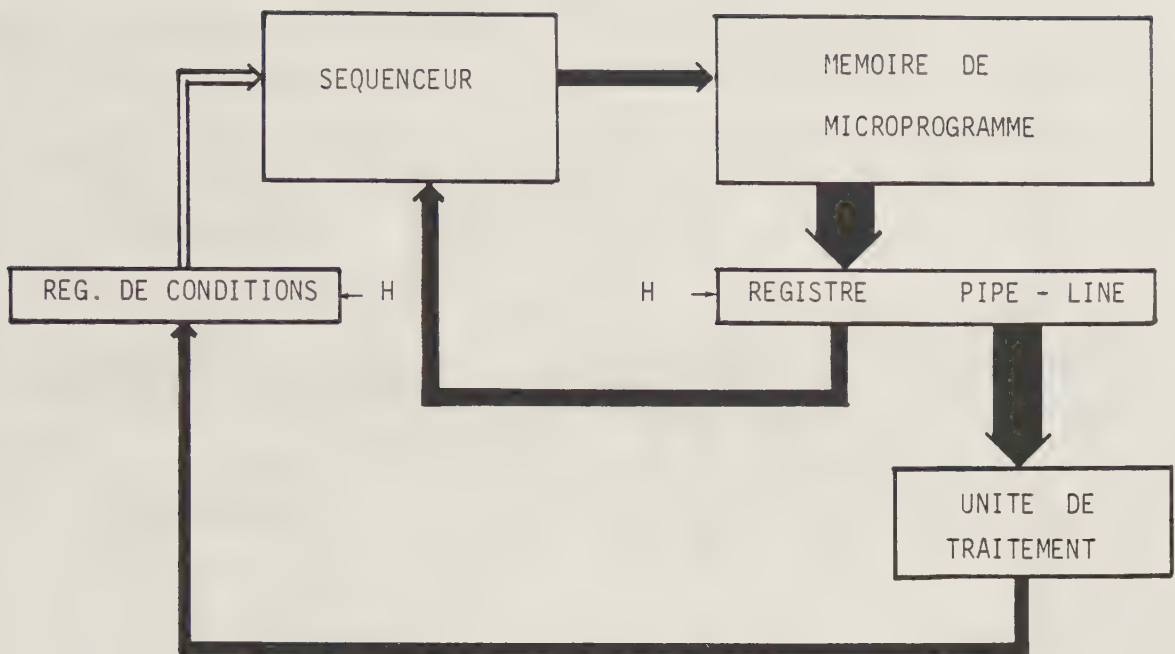


Figure 2 : Architecture Pipe-line

Les performances du calculateur peuvent être encore améliorées en adaptant la longueur du cycle à chaque microinstruction devant être exécutée. Cette fonction est obtenue par l'utilisation d'une horloge programmable commandée par la micro-instruction elle-même.

1.2. Calculateur vecteur (figure 3)

Le calculateur vecteur reprend l'architecture classique décrite précédemment mais dispose de plusieurs unités de traitement fonctionnant en parallèle. Chaque unité de traitement possède une unité arithmétique, des registres de travail, une mémoire et un coupleur. Les échanges de données sont effectués, à l'intérieur du système, sur un bus rapide. Une unité spécialisée assure l'adressage des mémoires et commande les transferts d'informations internes et externes.

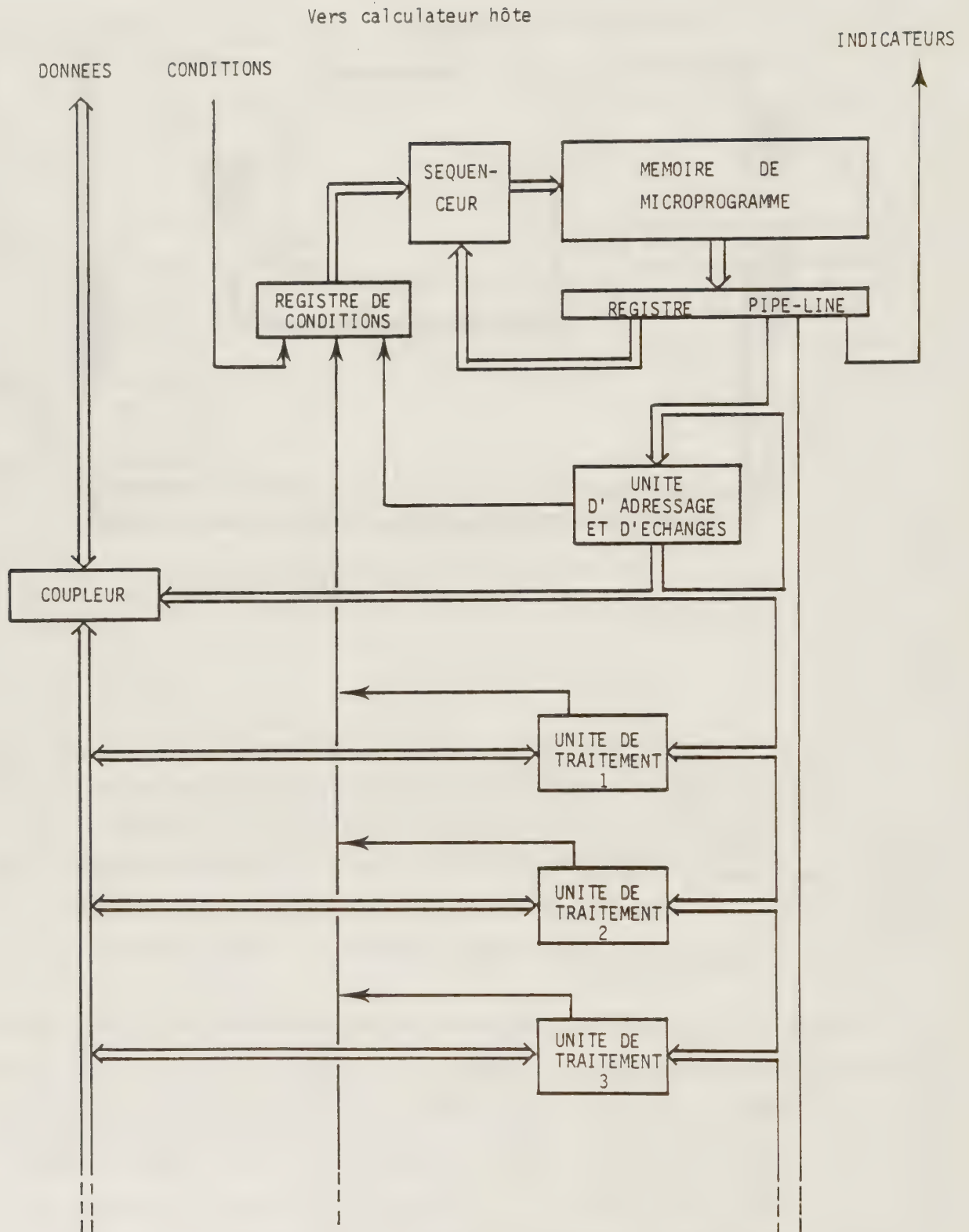


Figure 3 : Calculateur vecteur

Le calculateur vecteur est un outil de calcul performant mais est mal adapté à la commande directe d'un processus (acquisition, émission de données ; traitement des interruptions ...). Il est destiné au contraire à venir en aide à un calculateur-hôte en effectuant les traitements itératifs nécessaires au calcul de la commande. Les échanges avec le calculateur hôte s'effectuent en parallèle à travers un coupleur. Des signaux de contrôle sont émis par chaque calculateur pour la synchronisation des transferts de données.

Pendant la phase de mise au point du prototype le calculateur hôte est un système de développement facilitant la programmation et permettant de tester les différentes fonctions du calculateur. Il est complété par un pupitre de contrôle permettant l'exécution du programme en mode pas à pas avec visualisation de la microinstruction en cours d'exécution ainsi que de nombreux autres signaux.

Le calculateur que nous nous proposons de construire a son architecture simplifiée du fait de sa spécialisation pour la résolution d'un type de problème bien défini. Cela apparaît principalement dans le système d'adressage et dans l'unité de commande des échanges.

Une autre caractéristique du calculateur est sa modularité permettant son adaptation à la manipulation de vecteurs de tailles différentes. Pour le prototype nous nous sommes fixés une taille de une à seize unités de traitement, en prévoyant de réaliser une première tranche de huit unités.

2 - L'UNITE DE COMMANDE (figure 4)

L'unité de commande gère les unités de traitement et les échanges internes et externes. Elle comprend la mémoire contenant le microprogramme, le registre pipe-line mémorisant la microinstruction en cours d'exécution le séquenceur assurant le déroulement du microprogramme et une logique de commande des échanges.

2.1. La mémoire et le registre pipe-line

Dans la version définitive du calculateur la mémoire de micro-programme sera une mémoire morte. La vitesse d'exécution des micro-instructions dépendant du temps d'accès de la mémoire nous utiliserons des mémoires bipolaires programmables par claquage de fusibles. Pour la mise au point du prototype nous remplaçons la mémoire morte par une mémoire vive pouvant être chargée depuis le calculateur hôte [27]. Pour permettre le fonctionnement du calculateur à vitesse réelle nous employons des mémoires vives bipolaires. La taille de la mémoire est déterminée par la longueur du mot microinstruction et par la dimension du microprogramme soit au départ 256 mots de 64 bits. Lors de son chargement la mémoire est organisée en 1024 mots de 16 bits et connectée directement au bus du calculateur hôte.

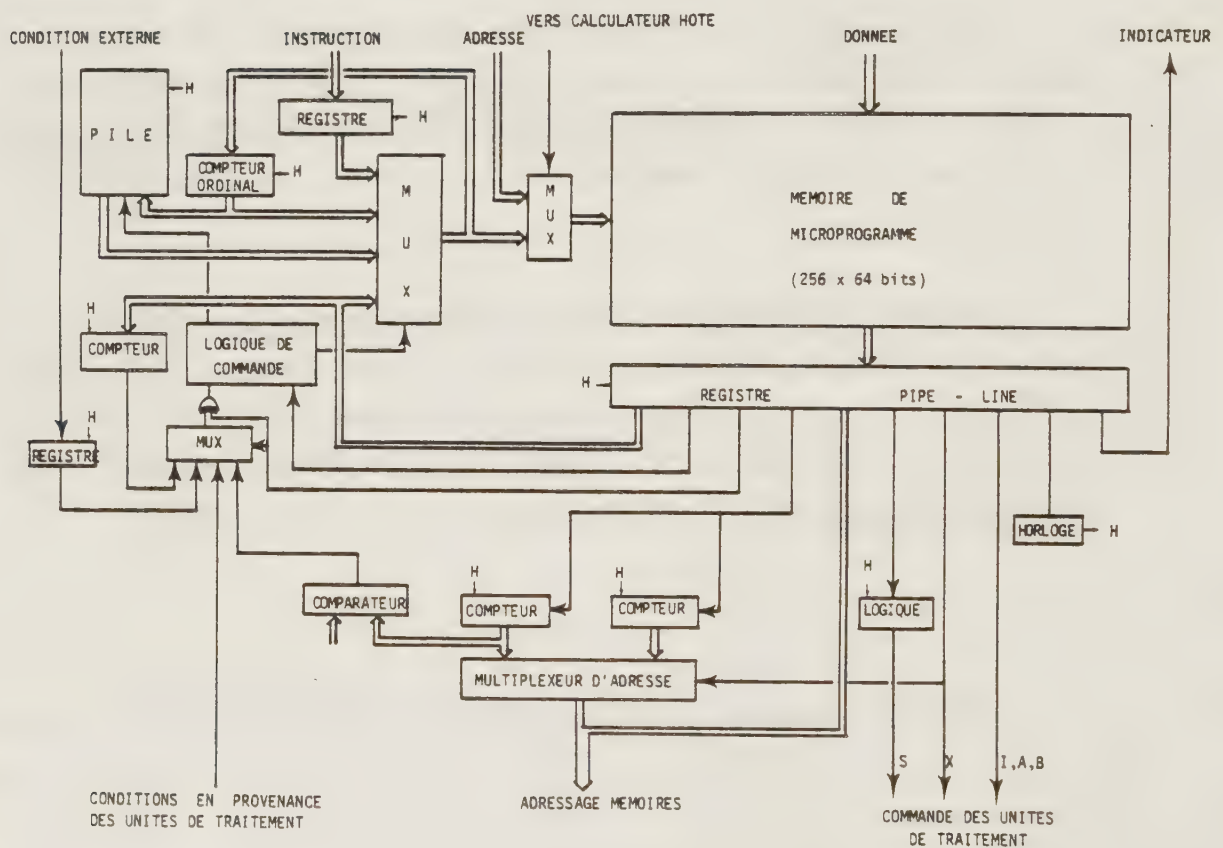


Figure 4 : Unité de commande

La microinstruction est chargée, pour être exécutée, dans le registre pipe-line. Elle se décompose en différents champs commandant chacun une fonction du calculateur (annexe V-1).

2.2. Le séquenceur

Le séquenceur commande le déroulement du microprogramme en fonction du champ "séquenceur" de la microinstruction et des conditions internes (unités de traitement, compteurs) et externes (calculateur hôte).

L'adresse de la prochaine microinstruction est sélectionnée par un multiplexeur et peut provenir de quatre sources :

- Le compteur ordinal ; il contient la dernière adresse émise par le séquenceur incrémentée de un et permet le déroulement séquentiel du microprogramme.
- Le registre pipe-line ; le séquenceur effectue un saut à l'adresse spécifiée dans la microinstruction (Adr.).
- La pile ; elle permet de mémoriser le contenu du compteur ordinal et de reprendre par la suite l'exécution du programme à l'adresse conservée lors d'un retour de sous-programme ou pour l'exécution d'une boucle.
- Le registre instruction ; il contient l'adresse de début d'un microprogramme chargée depuis le calculateur hôte (une logique de décodage de l'instruction n'est pas nécessaire ici).

Un multiplexeur sélectionne, sous le contrôle de la microinstruction (Cond.), la condition utilisée pour réaliser une rupture de séquence conditionnelle. Pour accroître la souplesse du séquenceur la sortie du multiplexeur peut être complémentée ou imposée.

Le séquenceur est complété par un compteur de boucle qui, chargé par la microinstruction (Adr.), autorise la répétition d'un groupe d'instructions un certain nombre de fois.

Une logique combinatoire commande les différentes fonctions du séquenceur à partir du champ "saut" de la microinstruction.

Le prototype utilise les circuits Am 2909 (séquenceurs 4 bits) ainsi que les circuits associés Am 29811 (logique de commande) et 169 (compteur synchrone) [30]. Pour une réalisation définitive l'emploi du séquenceur 12 bits Am 2910 assurera une meilleure intégration des fonctions ainsi que de meilleures performances au système.

2.3. Commande de l'adressage et des échanges

Si le séquenceur possède une architecture classique, au contraire l'unité d'adressage et d'échanges est adaptée au problème à résoudre. Nous allons retrouver ici les trois aspects du calculateur : traitement vectoriel, spécialisation et modularité.

Il est bon de rappeler le type de calcul devant être effectué par le calculateur. On peut le résumer par une équation de récurrence vectorielle :

$$(2.1) \quad z^{n+1} = M z^n \quad n = 0, \dots, N-1 \quad z^0 \text{ fixé}$$

Ce qui peut s'explicitier sous la forme de I équations :

$$(2.2) \quad z_i^{n+1} = \sum_{j=1}^I m_{ij} z_j^n$$

Chaque équation est résolue par une unité de traitement contenant dans sa mémoire la trajectoire $(z_i^0, \dots, z_i^n, \dots, z_i^N)$ et la ligne i de la matrice M soit $(m_{i1}, \dots, m_{ij}, \dots, m_{iI})$.

Le système d'adressage se décompose en deux parties :

- D'une part l'adressage en parallèle des mémoires de toutes les unités de traitement est nécessaire en vue de lire ou d'écrire un vecteur :

$$\begin{bmatrix} z_1^n \\ \vdots \\ z_I^n \end{bmatrix}$$

ou une colonne de matrice :

$$\begin{bmatrix} m_{1j} \\ \vdots \\ m_{Ij} \end{bmatrix}$$

Cette fonction est réalisée simplement en concaténant un champ de la microinstruction adressant une trajectoire ou une matrice avec le contenu d'un compteur indexant la composante. Un multiplexeur permet de sélectionner l'un des deux compteurs :

- compteur de pas contenant l'indice n ;
- compteur de composante contenant l'indice j .

Le débordement du compteur de pas est une condition utilisée pour tester la fin d'une phase de calcul à la fois en comptage (récurrence directe) et en décomptage (récurrence inverse).

- D'autre part il faut pouvoir adresser une unité de traitement j pour qu'elle fournisse par exemple aux autres unités i la composante z_j nécessaire au calcul des termes $m_{ij} z_j$. De même lors du chargement initial des mémoires elles seront successivement validées pour recevoir leurs données propres.

Nous utilisons pour cela deux registres à décalage répartis sur les unités de traitement (figure 5). Le premier sélectionne une seule unité de traitement (j) et permet la sortie des informations présentes sur le bus interne vers le bus principal. La donnée émise (D_j) peut alors être utilisée par les autres unités de traitement. Le compteur est ensuite incrémenté pour indexer l'unité de traitement suivante. Lorsque la dernière unité de traitement (I) est atteinte un signal est envoyé vers le multiplexeur de conditions afin de faire progresser l'exécution du microprogramme. De même le signal provenant d'une unité intermédiaire (J) peut également être pris en compte pour la manipulation de vecteurs de dimension plus faible (comme par exemple la commande u). Le deuxième registre, identique au premier, adresse une seule mémoire (i) pour y ranger une donnée (dans le cas de la figure 5 D_j).

Le nombre d'unités de traitement peut être augmenté sans qu'aucune modification ne soit effectuée sur le microprogramme. Il est suffisant en effet de rajouter des cartes unités de traitement (comprenant leurs portions de registres) puis d'établir les liaisons nécessaires (cavaliers) entre les registres et les lignes de conditions.

La microinstruction précise (Ech.) la nature des opérations à effectuer (écriture, lecture) ainsi que la provenance et la destination des données (mémoire, unité arithmétique, entrée/sortie).

2.4. Commande des autres fonctions

La commande des opérations de calcul est obtenue par des signaux directement disponibles à la sortie du registre pipe-line et envoyés vers les unités de traitement (unité arithmétique ...). Ce champ microinstruction sera détaillé dans le paragraphe suivant consacré à l'unité de traitement.

La microinstruction commande également l'horloge programmable. Celle-ci est réalisée à l'aide d'un décompteur chargé directement depuis la mémoire de microprogramme alors que la microinstruction est chargée dans le registre pipe-line.

La dernière fonction de l'unité de commande est la fourniture d'indications au calculateur-hôte pour la synchronisation des échanges et par la suite pour les alarmes .

3 - UNITE DE TRAITEMENT (figure 6)

Chaque unité de traitement se compose de deux parties :

- une unité de calcul comportant une unité arithmétique, des registres ainsi que des multiplexeurs permettant le transfert et le décalage des données ;
- une mémoire conservant les informations.

Une logique de commande des échanges gère l'accès au bus interne et la liaison avec le bus général du calculateur.

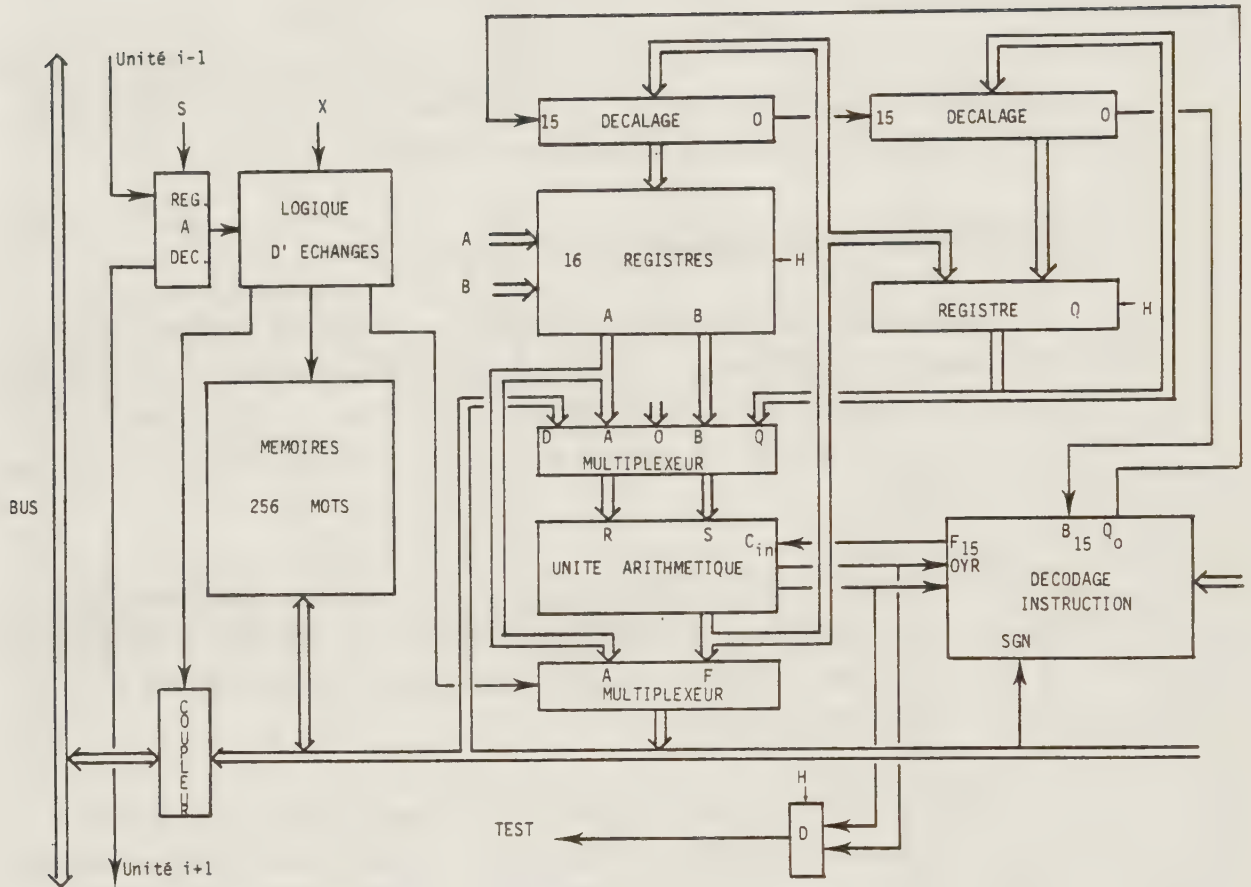


Figure 6 : Unité de traitement

3.1. L'unité de calcul

L'élément principal de l'unité de calcul est l'unité arithmétique effectuant les opérations de base (additions, soustraction). Elle reçoit les opérandes d'un multiplexeur par les entrées *R* et *S* et fournit le résultat par la sortie *F*.

Une mémoire de seize registres à double sortie A et B contient les données couramment utilisées. Elle peut être chargée par les données présentes en sortie de l'unité arithmétique après un décalage éventuel d'une position. Un registre supplémentaire Q est chargé depuis la sortie de l'unité arithmétique ou par lui-même avec dans ce cas une possibilité de décalage.

Un premier multiplexeur sélectionne les deux opérandes R et S parmi cinq sources :

- le registre adressé par A
- le registre adressé par B
- le registre Q
- le bus interne de l'unité de traitement
- zéro

Un deuxième multiplexeur commande la sortie des informations vers le bus interne de l'unité de traitement depuis deux sources :

- le registre adressé par A
- la sortie de l'unité arithmétique.

La réalisation de l'unité de calcul, à partir de quatre tranches de microprocesseur bipolaire de quatre bits Am 2901, ne fait appel qu'à un nombre limité de boîtiers.

Pour simplifier encore l'architecture nous ne conservons des possibilités de décalage que le décalage numérique à droite (division par 2) en double longueur (B et Q).

Nous ajoutons, en revanche, deux fonctions essentielles réalisées en parallèle sur toutes les unités de traitement :

- la multiplication microprogrammée ;
- le calcul de la valeur absolue .

Pour cette raison nous voyons apparaître en entrée du décodeur d'instruction (annexe V.2) des conditions utilisées pour réaliser ces opérations :

- F_{15} ; le signe en sortie de l'unité arithmétique
- Q_0 ; le bit de plus faible poids du registre Q lors d'un décalage
- OVR ; l'indicateur de débordement
- SGN ; le signe de la donnée présente sur le bus.

Réciproquement le décodeur d'instruction fournit les signaux :

- C_{IN} ; l'entrée de retenue
- B_{15} ; le bit de plus fort poids du registre B lors d'un décalage

Les indicateurs de signe et de dépassement (F_{15} et OVR) sont mémorisés sur chaque carte, pour faciliter la mise au point, puis transmis vers le multiplexeur de condition par deux lignes "Test" communes (fonction OU).

3.2. Rangement et transfert des données

La mémoire contient les données utilisées par l'unité de traitement soit les trajectoires (par exemple x_i^n) et les coefficients des matrices (par exemple a_{ij}). Chaque unité de traitement du prototype est munie de 256 mots de 16 bits de mémoire bipolaire.

La logique d'échange commande l'accès au bus interne, l'écriture en mémoire et la sortie de données sur le bus général. Le bus interne reçoit des informations de l'unité de calcul, de la mémoire ou du bus général par le coupleur en fonction de la microinstruction (Ech.) et de l'état du compteur de sélection. Inversement les données peuvent être écrites en mémoire ou communiquées sur le bus général suivant le contenu de la microinstruction et des compteurs.

4 - MISE EN ŒUVRE DU CALCULATEUR-VECTEUR

4.1. Le calculateur hôte

Le calculateur vecteur n'est pas un système autonome mais est destiné à venir en aide à un miniordinateur hôte. Le calculateur hôte fournit au calculateur vecteur des données, lance l'exécution du calcul puis reçoit le résultat en échange de nouvelles données. Il assure simultanément la gestion du processus et la communication avec les éléments extérieurs.

A l'initialisation du système, ou lors de la modification des paramètres, le calculateur vecteur est chargé du modèle du processus, du critère et des contraintes ($A, B, P, Q, U_{\max}, U_{\min}$) et éventuellement d'une première estimation de la trajectoire de commande $\{u^n\}$. Par la suite le calculateur vecteur reçoit, au début de chaque cycle, l'état initial x^0 et les trajectoires de couplage $\{e^n\}$ et $\{f^n\}$, il effectue le calcul puis remet la commande $\hat{u}^0$ et les trajectoires d'état et d'état adjoint $\{x^n\}$ et $\{\lambda^n\}$. Les échanges doivent pouvoir être réalisés sans trop ralentir le calculateur vecteur ni interrompre le calculateur hôte occupé à d'autres tâches.

L'utilisation d'une mémoire tampon accessible par les deux calculateurs permet de répondre à ces impératifs. La mémoire est normalement raccordée au calculateur-hôte. Celui-ci met à jour en permanence les trajectoires de couplage en fonction d'informations reçues des autres calculateurs. A chaque cycle il mesure l'état sur le processus puis le charge dans la mémoire tampon. La mémoire est alors laissée à l'usage du calculateur vecteur qui effectue en un temps très bref l'acquisition des données dont il a besoin puis l'écriture des résultats.

Il n'est pas dans notre propos d'étudier l'architecture du miniordinateur hôte, cependant il paraît souhaitable, pour la conduite décentralisée de grands systèmes, d'ajouter à un calculateur classique des processeurs spécialisés (figure 7). La gestion de la ligne de communication par un dispositif intelligent libère le processeur central de l'attente fastidieuse d'une donnée transmise épisodiquement et régularise le flux d'informations.

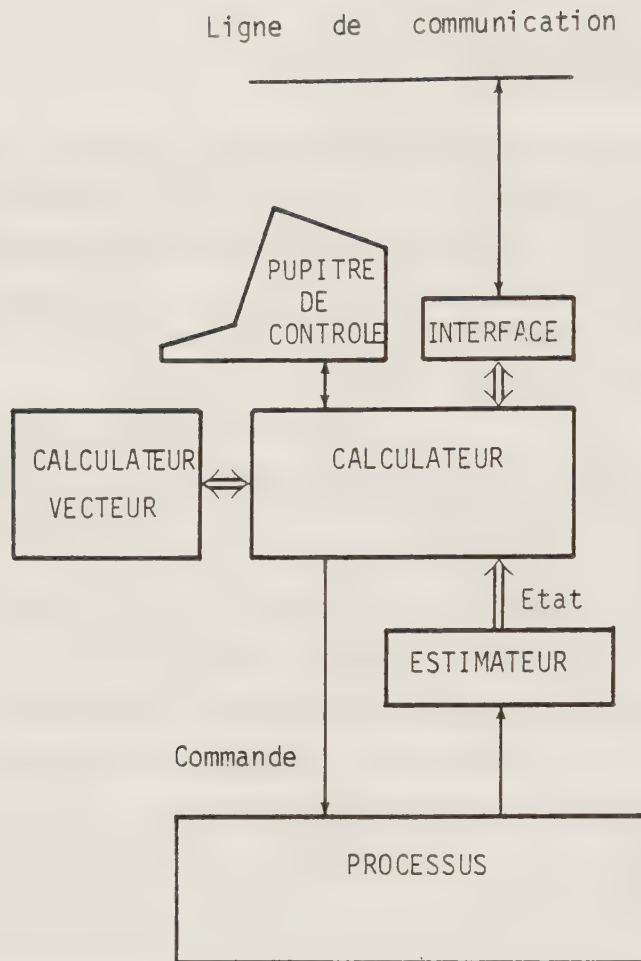


Figure 7 : Calculateur hôte et calculateurs périphériques

L'observation de l'état du processus physique n'est pas toujours une fonction simple et peut avantageusement être confiée à un autre processeur déterminant l'état à partir de mesures réalisées sur le processus. Le calculateur-hôte peut alors être réservé à la commande du calculateur vecteur mais aussi à d'autres tâches comme par exemple la gestion d'une interface homme-machine.

4.2. Le prototype

Pour la mise au point du calculateur vecteur il est inutile que le calculateur-hôte réponde à tous les impératifs énoncés dans le paragraphe précédent. Il doit, en revanche, assurer le chargement des programmes et le contrôle des différents éléments du calculateur vecteur.

Les échanges de données entre les deux machines n'ont pas besoin d'être performants, ils sont donc effectués, pour simplifier la réalisation, par un coupleur et synchronisés par des signaux d'accusé de réception. Les deux autres fonctions principales du calculateur-hôte sont le chargement de la mémoire vive de microprogramme et l'écriture dans le registre instruction. Cette dernière possibilité est utile pour lancer l'exécution d'un des microprogrammes contenus dans la mémoire, par exemple pour vider la mémoire de données entre deux calculs successifs. Pour faciliter la mise au point un dispositif de sécurité isole par des portes trois états, en état haute impédance, le calculateur-hôte lorsque l'alimentation du calculateur vecteur est coupée.

La mise en route du calculateur vecteur est assurée par un pupitre de commande comprenant les fonctions suivantes :

- initialisation ; remise à zéro du compteur ordinal et de certains autres compteurs, arrêt de l'horloge ; fonction réalisée automatiquement à la mise sous tension
- chargement ; la mémoire de commande est reliée au bus du calculateur-hôte pour le chargement du microprogramme
- marche ; mise en marche de l'horloge
- arrêt ; arrêt de l'horloge
- pas à pas ; l'horloge émet une seule impulsion.

Le pupitre visualise par ailleurs différents signaux :

- alimentation ; tension alimentation correcte
- adresse ; adresse de la prochaine microinstruction

- instruction ; microinstruction en cours d'exécution (67 bits)
- test ; conditions à l'entrée du multiplexeur
- compteur de pas ; 5 bits utilisés
- compteur de composante ; 4 bits
- compteur de boucle ; 4 bits

Un pupitre annexe permet le contrôle, sur chaque unité de traitement, du contenu des registres à décalage et des conditions mémorisées au niveau de chaque unité arithmétique.

ANNEXE V.1

LA MICROINSTRUCTION

* Champ séquençement

Le champ se compose de trois parties :

- commande du séquenceur
- sélection de la condition
- adresse de saut

Le séquenceur (2 x Am 2909) est commandé par un circuit spécialisé (Am 29811) placé en sortie du registre pipe-line. Nous utiliserons les instructions suivantes :

<u>INSTRUCTION</u>	<u>FONCTION</u>
JZ	Saut à l'adresse 0
CJS	Appel conditionnel d'un sous programme
CJP	Saut conditionnel
PUSH	Empilage et chargement conditionnel du compteur
JSRP	Appel d'un sous programme, l'adresse étant contenue soit dans le registre du séquenceur (condition fausse), soit dans le registre pipe-line (condition vraie)
JRP	Saut à l'adresse contenue dans l'un des deux registres
RFCT	Saut à l'adresse contenue dans la pile avec décrémentation du compteur, sortie de boucle si la condition est vraie (utilisation avec PUSH)
RPCT	Saut à l'adresse contenue dans le registre pipe-line avec décrémentation du compteur, sortie de boucle conditionnelle
CRTN	Retour conditionnel de sous-programme

CJPP	Saut conditionnel avec dépilage
LDCT	Chargement compteur
LOOP	Saut à l'adresse contenue dans la pile, sortie de la boucle conditionnelle (utilisation avec PUSH)
CONT	Déroulement séquentiel
JP	Saut

La condition peut être choisie parmi les huit entrées du multiplexeur et ensuite éventuellement complémentée :

<u>INSTRUCTION</u>	<u>FONCTION</u>	
F	Faux } Vrai }	exécution inconditionnelle d'instructions séquenceur
V		
NCS	Compteur séquenceur différent de zéro	
CS	Compteur séquenceur égal à zéro	
NCN	Fin de vecteur non atteinte	} suivant sélection câblée
CN	Fin de vecteur atteinte	
NCM	Fin de vecteur non atteinte	
CM	Fin de vecteur atteinte	
CP	Compteur de pas (temps) à zéro ou au maximum	
NCP	Compteur de pas à l'intérieur des limites	
H	Test du calculateur hôte positif	
NH	Test du calculateur hôte négatif	
NDP	Pas de dépassement	
DP	Dépassement de capacité (sur au moins une unité)	
POS	Toutes les valeurs calculées sont positives	
NPOS	Une valeur est négative	

L'adresse de saut est enfin donnée en hexadécimal.

* Champ indexation

Ce champ regroupe la commande de deux compteurs servant à l'adressage indexé :

- le compteur de pas contenant la variable temps
- le compteur de coefficients utilisé pour désigner une colonne dans une matrice

Les instructions sont les suivantes :

<u>INSTRUCTION</u>	<u>FONCTION</u>
NOP	Mémorisation de la valeur courante
LOAD	Chargement (remise à zéro)
INC	Incrémentation
DEC	Seulement pour le compteur de pas : décrémentation

Champ sélection

Ce champ assure la commande des registres permettant le choix d'une unité de calcul pour la lecture d'une donnée ou pour l'écriture en mémoire. Pour chaque registre le jeu d'instruction sera le même :

<u>INSTRUCTION</u>	<u>FONCTION</u>
NOP	Mémorisation
INIT	Remise à zéro (aucune unité n'est adressée)
INC	Sélection de l'unité suivante

* Champ U.A.L.

Ce champ commande les unités arithmétiques et logiques (Am 2901) et permet de sélectionner :

- la destination (rangement du résultat)
- l'opération effectuée
- les sources (contenant les opérandes)

Toutes les fonctions du 2901 peuvent être exécutées, mais il est aussi possible d'exécuter certaines fonctions particulières au problème à traiter grâce à la dernière partie de ce champ :

- sélection du deuxième jeu de fonctions (fonctions spéciales).

Le résultat d'une opération peut être rangé dans un registre avec ou sans décalage et sorti vers l'extérieur.

<u>INSTRUCTION</u>	<u>FONCTION</u>
QREG	Rangement dans le registre Q
NOP	Pas de rangement (sortie vers l'extérieur)
RAMA	Rangement dans le registre B, le registre A est sorti directement vers l'extérieur
RAMF	Rangement dans le registre B
RAMQD	Rangement dans B avec décalage de B et Q à droite (pour le câblage réalisé cette opération revient à effectuer un décalage numérique sur le mot de double longueur contenu dans BQ)
RAMD	Rangement dans B avec décalage à droite
RAMQU	} Non utilisé
RAMU	

Nous conservons la totalité des opérations arithmétiques et des fonctions logiques, ces dernières étant plus spécialement utilisées pour réaliser les fonctions spéciales :

<u>INSTRUCTION</u>	<u>FONCTION</u>
ADD	Addition $R + S$
SUBR	Soustraction $S - R$
SUBS	Soustraction $R - S$
OR	ou logique $R \vee S$
AND	et logique $R \wedge S$
NOTRS	Complément et et $\bar{R} \wedge S$
EXOR	ou exclusif $R \vee S$

Les opérandes (R et S) peuvent être choisis de huit façons différentes parmi les registres A, B et Q et l'entrée directe D :

<u>INSTRUCTION</u>	<u>FONCTION</u>	
AQ	R = A	S = Q
AB	R = A	S = B
ZQ	R = 0	S = Q
ZB	R = 0	S = B
ZA	R = 0	S = A
DA	R = D	S = A
DQ	R = D	S = Q
DZ	R = D	S = 0

Les fonctions spéciales sont sélectionnées par la dernière partie de ce champ :

<u>INSTRUCTION</u>	<u>FONCTION</u>
N	Jeu de fonction décrit ci-dessus
S	Fonctions spéciales câblées

* Champ échanges

Ce champ commande l'accès aux bus, l'écriture en mémoire et dans le registre de sortie. Il se compose de quatre parties :

- sélection de l'information émise sur les bus
- sélection du registre d'indexage
- commande de l'écriture en mémoire
- contrôle des entrées-sorties

L'information présentée sur le bus interne d'une unité de calcul peut provenir de l'unité arithmétique et logique, de la mémoire ou du bus général. Ce dernier reçoit des données d'une unité de calcul (sélectionnée par le registre de lecture) ou du système d'entrée-sortie.

INSTRUCTIONFONCTION

EALU	Si l'unité de calcul n'est pas sélectionnée elle reçoit des données de bus principal, si elle est sélectionnée son unité arithmétique est prise comme source.
ERAM	Idem pour la mémoire
ALU	L'information provient de l'unité arithmétique
RAM	L'information provient de la mémoire

Le registre utilisé pour l'adressage indexé est l'un des deux compteurs :

INSTRUCTIONFONCTION

C	Compteur de coefficients
P	Compteur de pas

L'écriture en mémoire peut être effectuée soit uniquement dans la mémoire sélectionnée par le registre d'écriture soit dans les mémoires :

INSTRUCTIONFONCTION

NOP	Pas d'écriture
WR	Ecriture dans toutes les mémoires
WRQ	Ecriture dans la mémoire sélectionnée

La sortie d'une information est réalisée en émettant celle-ci sur le bus principal en vue de la mémoriser dans le registre de sortie. L'entrée d'une information est obtenue par une lecture du registre d'entrée :

INSTRUCTIONFONCTION

NOP	Pas d'entrée ni de sortie
OUT	Sortie (bus principal → extérieur)
IN	Entrée (extérieur → bus principal)

* Champ adresse

Dans ce champ sont données en hexadécimal les adresses du registre A, du registre B et de la mémoire (poids forts).

* Champ indicateur

Ce champ est utilisé pour avertir le calculateur hôte de la fin d'un calcul (F).

* Champ horloge

Ce champ commande la durée du cycle d'horloge. Il contient le nombre de périodes d'horloge principale nécessaire à l'exécution de l'instruction en cours.

* Champ alarme

Ce champ sert à indiquer un fonctionnement anormal.

ANNEXE V.2

LE DECODEUR D'INSTRUCTION DE L'UNITE DE CALCUL

Nous conservons toutes les instructions du circuit Am 2901 bien que les instructions logiques ne soient pas indispensables. Nous ajoutons deux instructions supplémentaires pour implanter la multiplication micro-programmée et le calcul de la valeur absolue.

Notons :

- SGN ; le bit de plus fort poids sur le bus interne
- Q_0 ; le bit de plus faible poids en sortie du registre,
Q lors d'un décalage
- I ; l'instruction en sortie du registre pipe-line
- I(2901) ; l'instruction en entrée du 2901
- FS ; le bit instruction supplémentaire

Les équations reliant ces valeurs s'écrivent :

$$I_1(2901) = \overline{Q_0} \cdot I_1 + \overline{FS} \cdot I_1$$

$$I_4(2901) = \overline{SGN} \cdot I_4 + \overline{FS} \cdot I_4$$

Le reste de l'instruction restant inchangé.

Si $\overline{FS} = 0$ les instructions sont normalement exécutées. Au contraire si $\overline{FS} = 1$ l'exécution des instructions dépend de Q et de SGN de la façon suivante :

$$I_1 = 1 \quad , \quad I_4 = 0$$

$$I_1(2901) = \overline{Q_0}$$

ce qui permet la sélection des opérandes (A et B ou 0 et B) en fonction de Q_0 pour réaliser la multiplication.

$$I_1 = 0 \quad , \quad I_4 = 1$$

$$I_4(2901) = \overline{SGN}$$

l'opération effectuée (+A ou -A) est choisie en fonction du bit de signe SGN pour le calcul de la valeur absolue (ou de son opposé plus rapide à obtenir).

Le décodeur fournit également la retenue C_{IN} :

$$C_{IN} = I_3 + I_4$$

pour l'addition et la soustraction, ainsi que le bit de plus fort poids du registre B pour les décalages B_{15} :

$$B_{15} = OVR + F_{15}$$

pour le décalage numérique à droite et pour l'addition de deux mots de 16 bits avec résultat sur 17 bits intervenant lors de la multiplication.

VI - PROGRAMMATION

1 - CALCUL DE LA COMMANDE OPTIMALE

2 - MISE EN OEUVRE SUR LE CALCULATEUR VECTEUR

2.1. Calcul vectoriel

2.2. Multiplication et précision

2.3. Test de convergence et contraintes

3 - EXPLOITATION

3.1. La programmation

3.2. Exploitation expérimentale du calculateur

3.3. Mise au point du calculateur

Après avoir précisé la méthode de calcul utilisée nous décrivons plus précisément sa mise en oeuvre sur le calculateur vecteur. Nous terminerons par les moyens utilisés pour l'écriture et la mise au point des programmes ainsi que pour l'exploitation expérimentale du calculateur vecteur.

1 - CALCUL DE LA COMMANDE OPTIMALE

Le fonctionnement typique du calculateur vecteur comprend quatre phases (figure 1) :

- Acquisition des coefficients en provenance du calculateur-hôte (matrices, contraintes, test de convergence)
- Acquisition de l'état initial x^0 (et des trajectoires de couplage $\{e_i^n\}$ et $\{f_i^n\}$)
- Calcul de la commande $\hat{u}^0$
- Sortie de la commande $\hat{u}^0$ vers le calculateur hôte.

Les trois dernières opérations sont répétées pour permettre la conduite en boucle fermée d'un processus.

Le calcul est effectué suivant la méthode de relaxation exposée précédemment (IV,1.1). Les calculs à effectuer à chaque itération sont les suivants :

- Calcul de la trajectoire d'état :

$$(3.1) \quad x_i^{n+1} = \sum_{j=1}^I a_{ij} x_j^n + \sum_{j=1}^J b_{ij} u_j^n + e_i^n \quad n = 0, \dots, N-1$$

- Calcul de la trajectoire d'état adjoint :

$$(3.2) \quad \lambda_i^{n+1} = p_{ii} x_i^n + \sum_{j=1}^I a_{ji} \lambda_j^n + f_i^n \quad n = N, \dots, 1$$

(Nous supposons la matrice P diagonale pour éviter de prendre en compte des termes généralement nuls).

- Calcul de la commande :

- Calcul de v_i^n :

$$(3.3) \quad v_i^n = \sum_{j=1}^I r_{ij} \lambda_j^n \quad i = 1, \dots, J \quad n = 0, \dots, N$$

(Nous introduisons une matrice $R = -Q^{-1}BT$)

- Introduction du paramètre de relaxation :

soit $\{u_i^{n,-}\}$ la loi de commande calculée à l'itération précédente, nous calculons $G_\omega(U)$, (I,4.3)

$$(3.4) \quad v_{\omega,i}^n = \omega v_i^n + (1-\omega) u_i^{n,-} \quad i = 1, \dots, J \quad n = 0, \dots, N$$

- Application des contraintes

$$(3.5) \quad \begin{aligned} u_i^n &= v_{\omega,i}^n & \text{si} & \quad u_{i \min} \leq v_{\omega,i}^n \leq u_{i \max} \\ u_i^n &= u_{i \max} & \text{si} & \quad v_{\omega,i}^n > u_{i \max} \\ u_i^n &= u_{i \min} & \text{si} & \quad v_{\omega,i}^n < u_{i \min} \end{aligned}$$

- Test de convergence :

Si l'on désire s'assurer de la convergence du calcul le test doit théoriquement porter sur l'ensemble de la trajectoire $\{u_i^n\}$. Pratiquement, le problème à résoudre étant connu par avance et des essais préalables ayant été effectués, nous nous contentons d'arrêter le calcul lorsque la solution du problème u° ne varie presque plus d'une itération à l'autre c'est-à-dire quand :

$$||u^{o,-} - u^\circ|| \leq \delta$$

Compte-tenu des particularités du calculateur vecteur nous utilisons la norme :

$$||z|| = \text{Sup} (|z_i|)$$

En fonction du résultat du test le calcul sera réitéré ou bien le calculateur vecteur émettra la commande puis recevra une nouvelle valeur de l'état initial pour effectuer un nouveau calcul. Les opérations effectuées sur une unité de traitement pour le calcul de la commande apparaissent sur l'organigramme (figure 2).

2 - MISE EN ŒUVRE SUR LE CALCULATEUR VECTEUR

Nous allons nous limiter à la description de quelques particularités du programme dues à son implantation sur le calculateur vecteur.

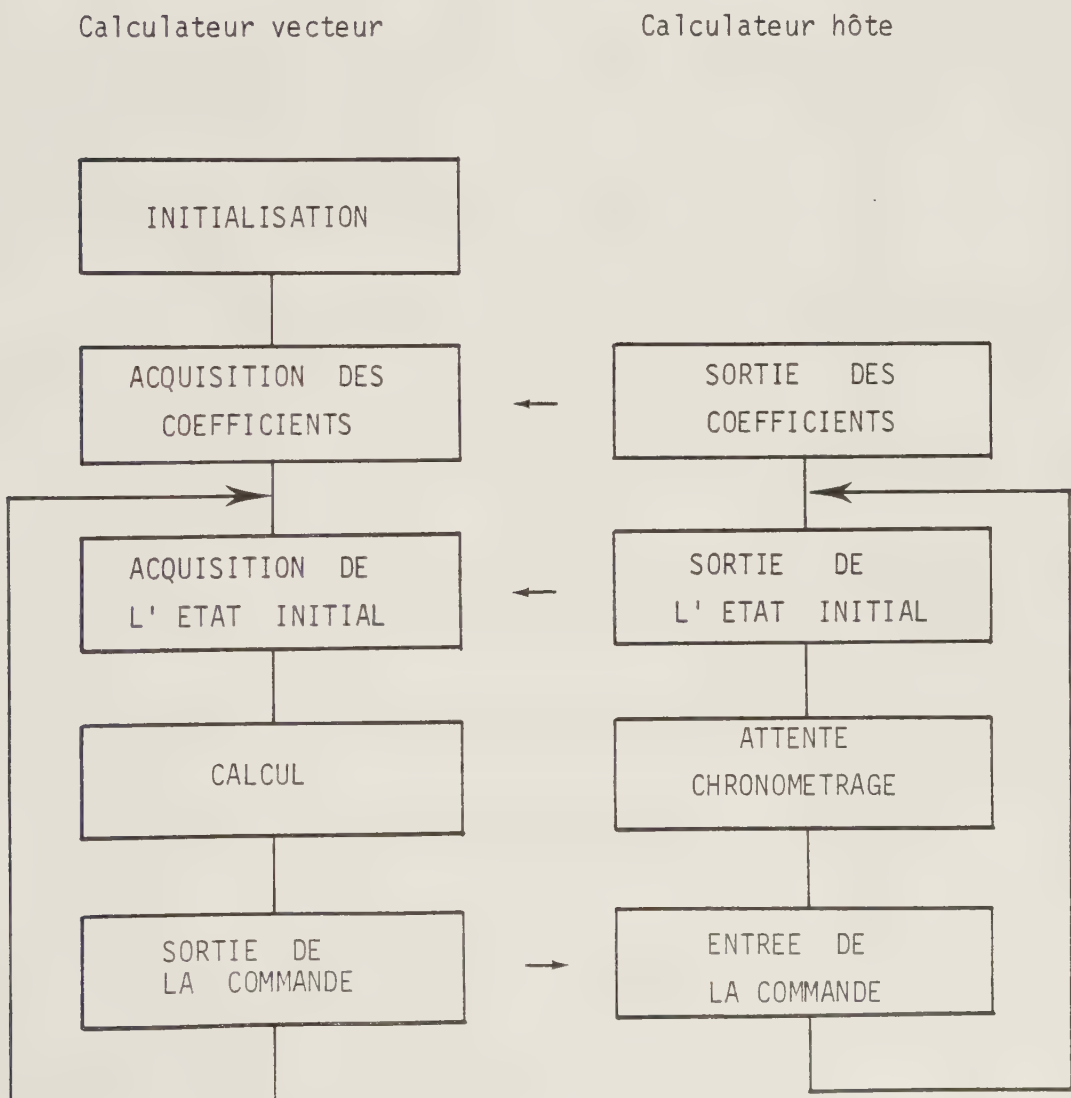


Figure 1 : Echanges entre le calculateur vecteur et le calculateur hôte.

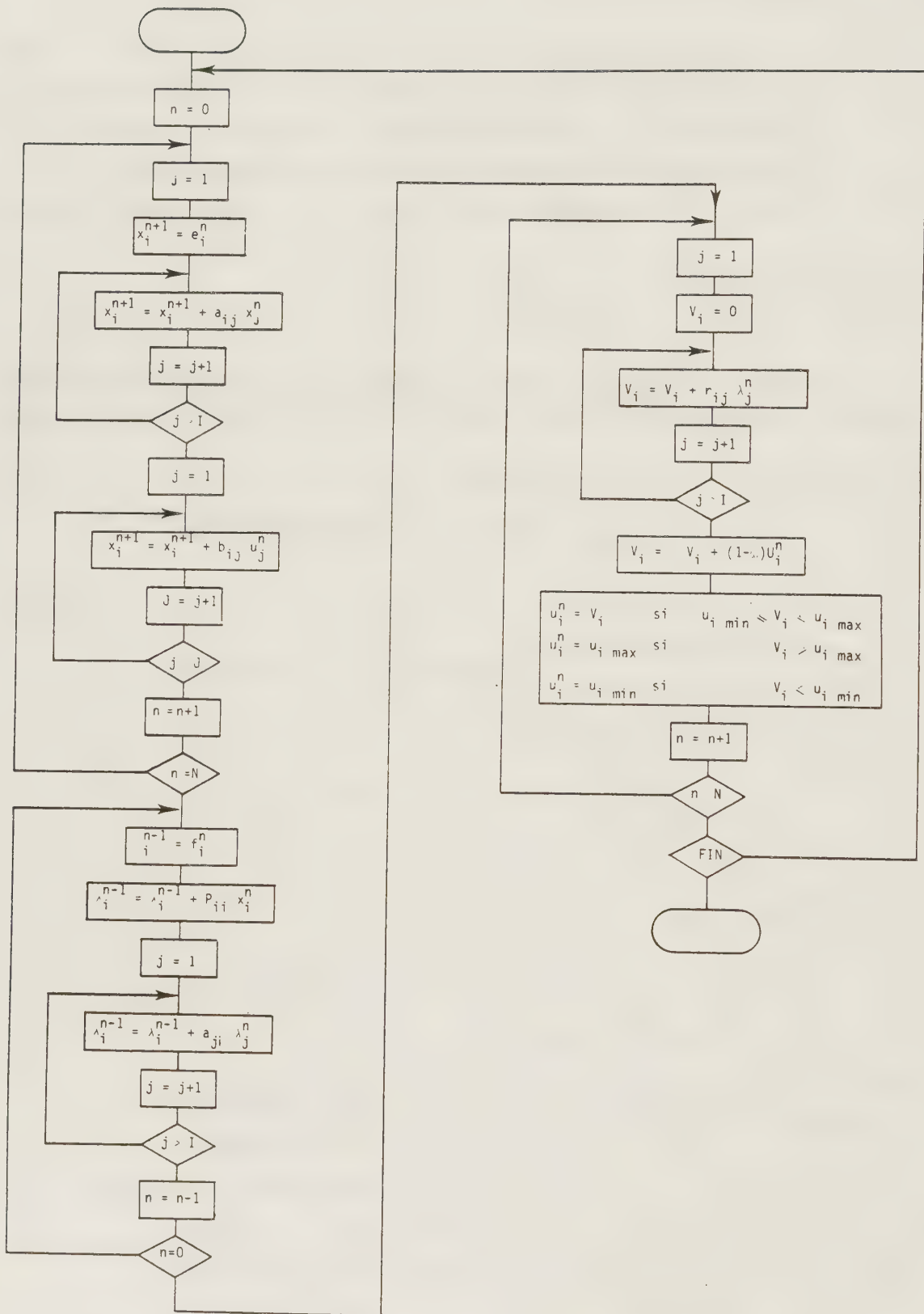


Figure 2 : Organigramme (opérations effectuées sur l'unité de traitement i)

2.1. Calcul vectoriel

Les différentes opérations sont effectuées en utilisant au maximum les possibilités du calculateur. Un calcul du type :

$$x_i^{n+1} = x_i^{n+1} + a_{ij} x_j^n \quad j \text{ fixé, } i = 1, \dots, I$$

est réalisé de façon classique en calculant :

$$\begin{aligned} x_1^{n+1} &= x_1^{n+1} + a_{1j} x_j^n && \text{sur l'unité de traitement 1} \\ &\vdots \\ x_I^{n+1} &= x_I^{n+1} + a_{Ij} x_j^n && \text{sur l'unité de traitement I} \end{aligned}$$

Cependant si l'on est amené à traiter des vecteurs de dimension plus faible comme la commande il est possible d'adopter une méthode plus performante, soit :

$$v_i = \sum_{j=1}^I r_{ij} \lambda_j^n \quad i = 1, \dots, J$$

Si nous effectuons le calcul de façon classique le temps nécessaire sera de (figure 2) :

$$I (T_a + T_m)$$

T_a = Temps nécessaire pour effectuer une addition

T_m = Temps nécessaire pour effectuer une multiplication

Au contraire si les produits sont effectués de la manière suivante

$$\begin{aligned} r_{i1} \cdot \lambda_1 & \quad \text{sur l'unité de traitement 1} \\ & \vdots \\ r_{iI} \cdot \lambda_I & \quad \text{sur l'unité de traitement I} \end{aligned}$$

on obtient chaque v_i en effectuant la somme :

$$v_i = \sum_{j=1}^I (r_{ij} \lambda_j^n)$$

cette opération étant répétée J fois le temps de calcul total est alors de :

$$J (I \cdot T_a + T_m)$$

Il est plus avantageux d'utiliser la seconde méthode si :

$$(3.6) \quad J < I \frac{(T_a + T_m)}{(I \cdot T_a + T_m)}$$

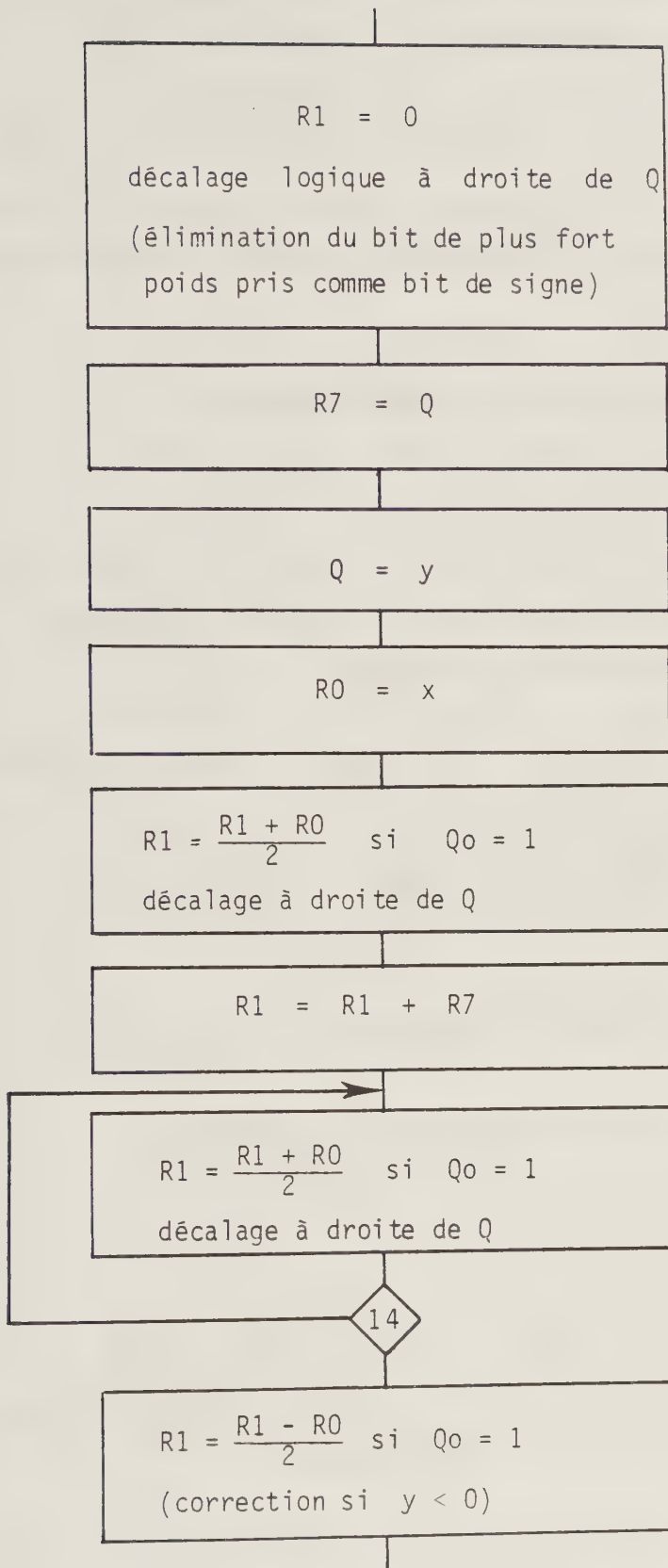
Comme la multiplication microprogrammée est beaucoup plus longue à effectuer que l'addition ($T_m \gg T_a$) la relation (3.6) sera assez souvent vérifiée.

En fait le choix de l'une ou l'autre méthode est plus complexe car les calculs ne sont pas réalisés dans les deux cas avec la même précision comme nous allons le voir dans le paragraphe suivant.

2.2. Multiplication et précision

Les erreurs de calcul proviennent des troncatures effectuées après les multiplications de mots de 16 bits. Les résultats étant rangés dans des mots de même longueur. Il est possible d'éviter l'accumulation de ce type d'erreur en conservant au niveau de chaque unité de calcul les 16 bits de poids faible du résultat d'une multiplication. Lors d'une somme de produits les 16 bits de poids faible conservés après une multiplication sont ajoutés au résultat de la multiplication suivante.

La multiplication avec signe (figure 3) comprend donc une phase supplémentaire destinée à ajouter au mot de poids faible du résultat le mot de poids faible du résultat précédent considéré comme un nombre entier sans signe.



Q contient le mot de poids faible du résultat de la dernière multiplication ;
 $R1$ contient le résultat de la multiplication.

Figure 3 : MULTIPLICATION $x.y$

2.3. Test de convergence et contraintes

L'opération fondamentale pour le test de convergence ainsi que pour l'application des contraintes est le calcul de la valeur absolue d'un nombre. Cette opération est câblée au niveau de chaque unité et peut être exécutée en parallèle sur toutes les composantes d'un vecteur.

Le test de convergence est réalisé en calculant :

$$\delta = |u_i^o - u_i^{o-}| \quad i = 1, \dots, J$$

en parallèle sur l'ensemble des unités de calcul. Le calcul est terminé lorsque tous les résultats sont positifs (la fonction "et" étant réalisée par la ligne de test).

Le calcul de u^n à partir de v^n est obtenu par un microprogramme à l'aide de la fonction valeur absolue :

$$u_i^n = \frac{\frac{v_i + u_{i \max}}{2} - \left| \frac{v_i - u_{i \max}}{2} \right| + u_{i \min}}{2} + \left| \frac{\frac{v_i + u_{i \max}}{2} - \left| \frac{v_i - u_{i \max}}{2} \right| - u_{i \min}}{2} \right| \quad i = 1, \dots, J$$

les opérations étant bien sûr effectuées en parallèle.

3 - EXPLOITATION DU CALCULATEUR VECTEUR

Nous allons décrire les moyens mis en oeuvre pour écrire les microprogrammes pour exploiter les résultats des calculs effectués sur le calculateur vecteur et enfin pour tester les différents éléments matériels du calculateur et mettre au point les microprogrammes.

3.1. La programmation

Dans sa version définitive le calculateur vecteur effectuera toujours le même calcul comandé par le microprogramme placé dans

une mémoire morte (du type PROM bipolaire). Le programme devra être optimisé et par conséquent écrit directement en langage microassembleur. Nous n'envisageons donc pas l'utilisation d'un langage plus évolué pour ce calculateur.

L'écriture manuelle d'un programme en langage machine est une source d'erreur trop importante (mot instruction de 64 bits).

Nous avons donc écrit un microassembleur implanté sur un système de développement pour microprocesseurs classiques (*). Bien que simplifié à l'extrême il permet à partir d'un fichier source, créé par l'éditeur du système, d'élaborer le code objet et de le ranger sur un autre fichier ainsi que l'impression d'un listing. D'autre part l'assembleur se décompose en une partie programme figée et une partie contenant les informations spécifiques à chaque architecture (mnémoniques, dimension des champs, position des constantes), ce qui permet de l'adapter rapidement à d'autres architectures.

Pour la mise au point du programme nous utilisons également un programme de chargement du code objet dans la mémoire de commande du calculateur vecteur.

3.2. Exploitation expérimentale du calculateur

Pour l'évaluation des performances du calculateur vecteur le système de développement lui fournit des valeurs puis attend les résultats en mesurant le temps de calcul (figure 2). Pour faciliter l'utilisation du calculateur vecteur le programme implanté sur le calculateur hôte est écrit dans le langage évolué disponible : Un Basic interprété. Bien que peu performant ce langage permet d'écrire facilement les programmes de test simples nécessaires aux premières expériences. Le programme de base permet les fonctions suivantes :

- Acquisition et modification des paramètres depuis une console.
- Impression des paramètres.

(*) PACE Development System, National Semiconductor

- Ecriture et lecture sur disque souple des paramètres
- Calcul de la loi de commande optimale, sur le système de développement, en virgule flottante
- Calcul de la loi de commande optimale, sur le système de développement, en virgule fixe
- Chargement de nouveaux paramètres dans le calculateur vecteur et lancement d'un calcul. Lorsque le calculateur vecteur fournit le résultat le système imprime le temps de calcul puis la commande $\hat{u}^0$. L'opérateur peut alors introduire une nouvelle valeur de l'état initial et relancer le calcul.

Ce programme est destiné à mesurer les performances du calculateur vecteur (temps de calcul, précision) en fonction de différents paramètres (paramètre de relaxation ω , test de convergence δ). A partir des fonctions précédentes il est simple d'écrire un programme pour étudier un aspect particulier du fonctionnement du calculateur vecteur.

3.3. Mise au point du calculateur

Pour mettre au point la partie matérielle du calculateur nous avons écrit des programmes de test qui seront conservés pour un contrôle ultérieur des éléments du calculateur. Ils permettent principalement de tester les ensembles suivants :

- la mémoire de commande
- le séquenceur et les différents compteurs
- les mémoires contenant les données
- les unités arithmétiques.

Certains éléments (comme la mémoire de commande) ne sont pas accessibles depuis le système de développement ; les résultats des tests sont obtenus grâce à un analyseur logique (*) connecté

(*) LAM 4850, Dolch

sur les différentes cartes en fonction des circuits à tester. La version définitive sera munie d'un programme de test automatique pour détecter une panne éventuelle et situer approximativement la partie défectueuse.

La mise au point des programmes est facilitée par la possibilité d'examiner après l'exécution d'un programme les registres et les mémoires. Nous utilisons à cet effet un programme de vidage des registres et des mémoires du calculateur vecteur dans celle du système de développement ; il est alors possible de les examiner et éventuellement de les imprimer. Réciproquement les mémoires du calculateur-vecteur peuvent être chargées depuis celle du système ou encore depuis un fichier rangé sur disque souple avec si nécessaire des modifications apportées par l'opérateur. Le fichier utilisé contient par exemple les paramètres rangés par le programme décrit au paragraphe précédent (3.2).

Le microprogramme à tester est placé entre les programmes de chargement et de vidage mémoire donnant ainsi à l'utilisateur la vision du traitement effectué. Une observation à l'analyseur logique des transferts d'informations fournit un état détaillé des opérations réalisées au cours d'un calcul.

CONCLUSION

Nous avons jusqu'à présent décrit les principes de fonctionnement du calculateur vecteur sans aborder les problèmes technologiques rencontrés et leur incidence sur les performances du système. Il est maintenant temps de combler cette lacune en étudiant la réalisation pratique du calculateur. Nous mesurerons ensuite les performances réelles du prototype et nous évaluerons celles du système définitif pour différentes configurations.

ANNEXE VI . 1

PROGRAMME DE CALCUL

Ce programme permet de calculer la commande optimale $\hat{u}^0$ pour un problème linéaire avec contraintes en utilisant un paramètre de relaxation. Dès que le calculateur est initialisé il attend des données du calculateur hôte, les charge dans sa mémoire et commence un calcul. A la fin du calcul il avertit le calculateur hôte (indicateur) puis transfère la commande $\hat{u}^0$. Il reçoit immédiatement un nouvel état initial x^0 et recommence un calcul.

Initialisation des compteurs

- CC : Compteur de coefficient ; il est utilisé pour adresser un élément dans une ligne de matrice (indice j)

- CP : Compteur de pas ; il est utilisé pour adresser un élément d'une trajectoire (indice n)

- CS : Compteur séquenceur ; il est utilisé pour compter le nombre de boucles exécutées

- CE : Compteur écriture ; il sélectionne l'unité de calcul dans la mémoire de laquelle on désire écrire

- CL : Compteur lecture ; il sélectionne l'unité de calcul en utilisant le bus pour transmettre une donnée.

Conditions de rupture de séquence

- H : Accès du calculateur hôte au calculateur vecteur

- CS : Compteur séquenceur à zéro

- CP : Compteur de pas à zéro ou au maximum fixé par les interrupteurs situés sur la carte unité de commande
- CN : Compteur d'écriture ou de lecture ayant atteint la valeur maximale I c'est-à-dire la dimension du vecteur d'état
- CM : Un des compteur a atteint J la dimension du vecteur de commande.

La lettre N placée devant une condition indique que le saut est réalisé lorsque la condition inverse se produit.

Utilisation des registres

- R0 à R7 : usage général
- R8 : P_i
- R9 : $u_{\max}$
- RA : $u_{\min}$
- RB : δ
- RC : ω

Utilisation de la mémoire

Chaque mémoire comprend 256 mots répartis de la façon suivante :

- 2 x 16 mots servant à la duplication des registres lors de la mise au point, l'adressage s'effectue en 0 et 1 le compteur CC étant utilisé comme index,
- 5 x 32 mots pour le rangement des trajectoires indexés par le compteur CP :

- | | | | |
|---|---|-----------------|--------------|
| 1 | : | $\{x^n\}$ | |
| 2 | : | $\{\lambda^n\}$ | |
| 3 | : | $\{u^n\}$ | |
| 4 | : | $\{e_i^n\}$ | } en réserve |
| 5 | : | $\{f_i^n\}$ | |

- 4 x 16 mots pour le rangement des lignes des matrices indexés par le compteur CC :

C : A

D : B

E : A^T

F : $-Q^{-1}B^T$

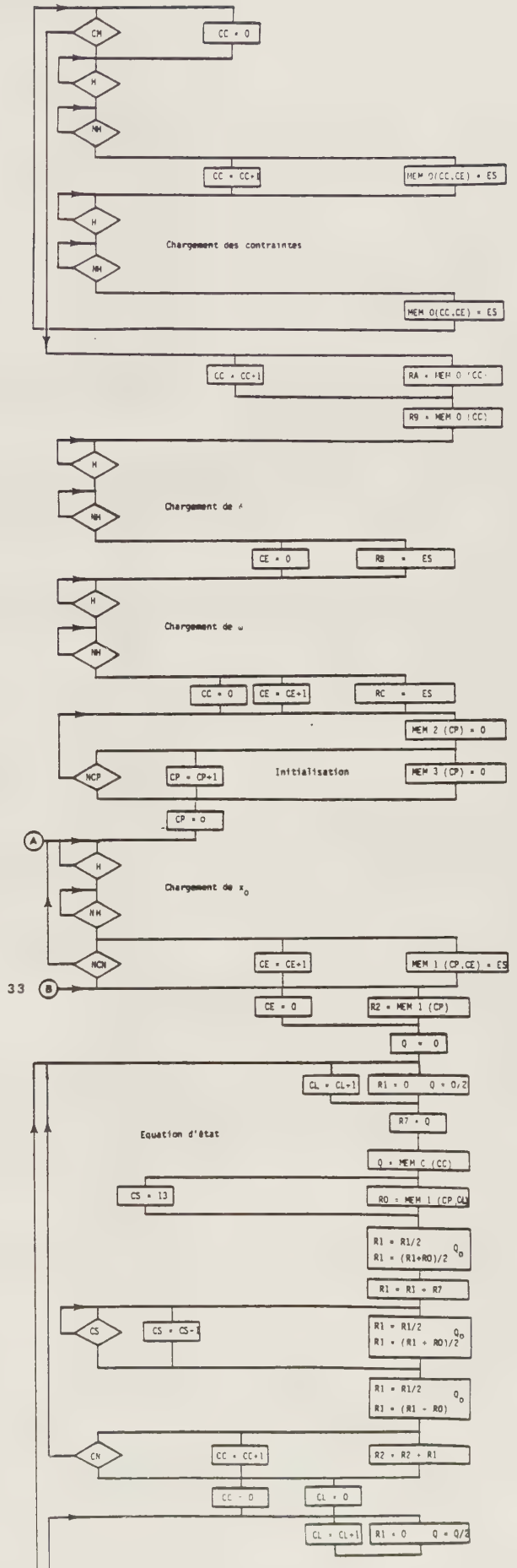
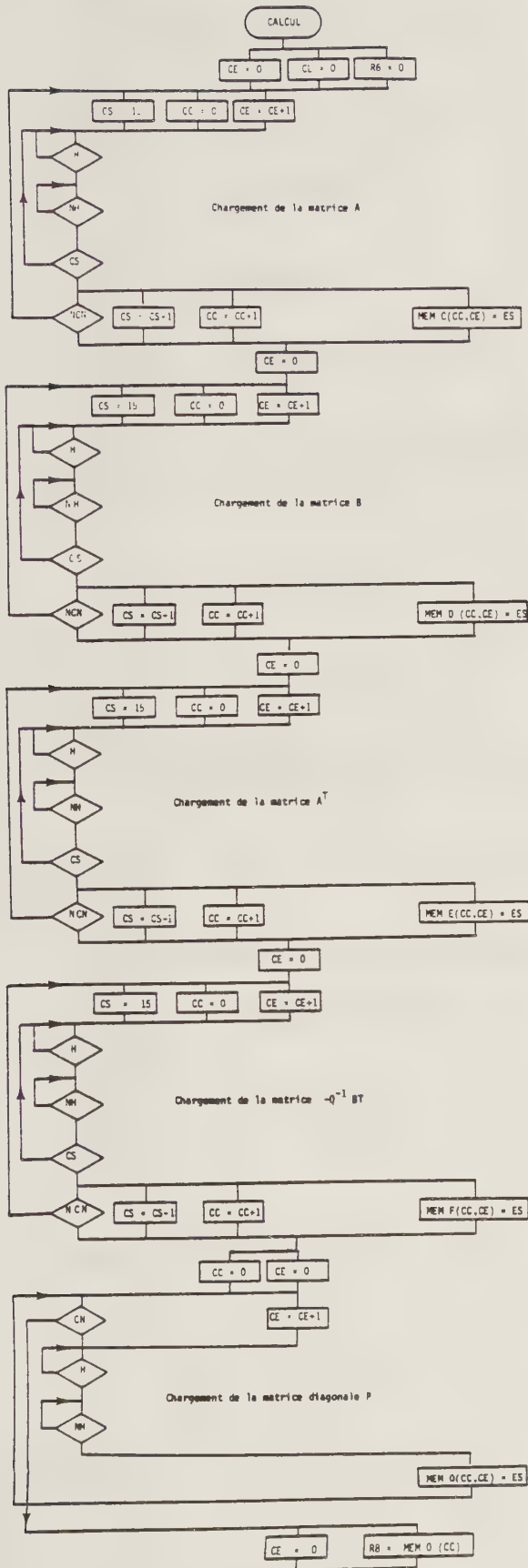
Notations

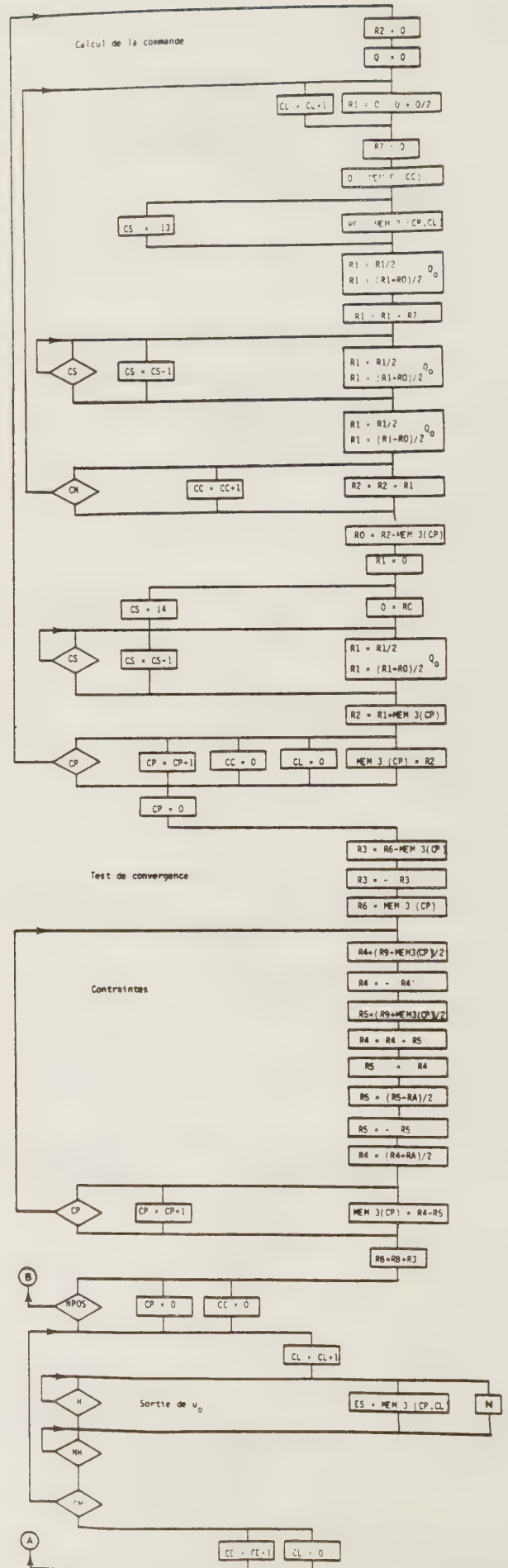
MEM C (CC, CE) = ES indique que l'écriture est effectuée dans la mémoire de l'unité de calcul sélectionnée par le compteur CE à l'adresse de la matrice A(C) dans l'élément indexé par le compteur CC, la source étant le coupleur d'entrée sortie.

Indicateur

L'indicateur actif à zéro est noté N.

00





VII - PERFORMANCES

1 - REALISATION DU PROTOTYPE

1.1. Choix des composants

1.2. Le câblage

2 - PERFORMANCES

2.1. Evaluation de la vitesse de calcul

2.1.1. Nombre d'itérations

2.1.2. Nombre d'instructions par itérations

2.1.3. Durée du cycle

2.2. Performances réelles du calculateur actuellement réalisé

2.2.1. Vitesse

2.2.2. Précision

2.3. Performances du calculateur complet

2.4. Amélioration des performances

3 - CONDUITE D'UNE SIMULATION DE PROCESSUS

3.1. Conduite centralisée

3.1.1. Commande d'un processus parfait

3.1.2. Régulation d'un processus perturbé

3.1.3. Commande d'un processus instable

3.2. Simulation de conduite décentralisée

CONCLUSION

Nous consacrons le début de ce chapitre à la description de la réalisation pratique du prototype du calculateur ainsi qu'à son expérimentation. Nous recherchons ensuite les solutions permettant l'amélioration éventuelle des performances.

1 - REALISATION DU CALCULATEUR

Nous abordons ici les aspects technologiques du calculateur comprenant le choix des composants utilisés et le câblage.

1.1. Choix des composants

Le choix de la technologie dépend de nombreux éléments :

- les performances recherchées
- l'immunité aux parasites
- le coût des composants
- la consommation
- la facilité de mise en oeuvre
- la disponibilité des composants ...

Nous avons effectué un compromis entre ces différents impératifs en accordant une grande importance aux deux derniers pour que la mise au point ne soit pas rendue trop laborieuse par des problèmes purement matériels. Nous utilisons donc la technologie TTL, dans laquelle nous trouvons la plupart des microprocesseurs en tranches ainsi que des mémoires rapides et une vaste gamme de circuits moins complexes.

Le choix de la famille 2900 [28] est lié bien sûr à l'architecture des éléments mais aussi à la disponibilité des produits. Le prototype est équipé de la version la moins rapide soit le séquenceur Am 2909 et l'unité de traitement Am 2901.

Le choix des mémoires est beaucoup moins évident car la technologie bipolaire est source d'une grande consommation et des inconvénients qui en découlent. Nous avons donc été tentés d'utiliser des

mémoires MOS compatible TTL au prix d'une diminution des performances. La mémoire de commande est un élément provisoire puisqu'elle sera remplacée par une mémoire morte. Cependant, afin de tester l'ensemble du calculateur à sa vitesse réelle de fonctionnement, nous avons jugé préférable l'emploi d'une mémoire bipolaire rapide compatible TTL (Am 29721). Les mémoires contenant les données sont en fait peu sollicitées et l'emploi de mémoires plus lentes ralentirait peu l'exécution du microprogramme grâce au cycle d'horloge variable (seule quelques micro-instructions demanderont un temps d'exécution plus long). La capacité de cette mémoire étant relativement faible la mise en oeuvre de mémoires TTL (93422) n'a pas une grande incidence sur la consommation ou le coût du calculateur tout en permettant un accroissement sensible des performances.

Les autres fonctions sont obtenues par l'utilisation de circuits intégrés TTL série LS dont le principal avantage est la réduction de la consommation. Nous n'avons pas essayé de réduire au minimum le nombre de boîtiers mais nous avons choisi des composants courants réalisant chacun une fonction élémentaire ce qui présente deux avantages :

- l'approvisionnement est grandement simplifié
- le remplacement par des éléments plus performants en vitesse (par exemple série FAST [29]) ou en sortance (par exemple buffer série 25 LS [30]) est immédiat.

Pour une version définitive les impératifs ne seraient pas les mêmes et une intégration plus poussée des différentes fonctions contribuerait largement à une simplification du câblage tout en améliorant les performances du calculateur.

1.2. Le câblage

Le calculateur vecteur comprend plusieurs éléments :

- des cartes supportant 1 ou 2 unités de traitement
- une carte unité de commande
- une carte d'interface avec le calculateur hôte
- un pupitre de commande.

Les cartes "unité de traitement" ont à quelques détails près leur forme définitive mais il n'en est pas de même de la carte "unité de commande" qui sera modifiée lors du remplacement de la mémoire vive par une mémoire morte et de la carte d'interface qui sera adaptée au calculateur hôte. Nous utilisons par conséquent une méthode de câblage permettant de réaliser des cartes à l'unité et d'effectuer facilement de légères modifications pour la mise au point du prototype. La technique de connexion par "wrapping" autorise l'implantation d'un grand nombre de composants sur une surface réduite (une centaine de boîtiers pour la carte "unité de commande"). Le câblage est effectué sur des cartes au format européen double percées au pas de 2,54 donnant toute liberté pour la disposition des éléments. Cette technique rend, en revanche, difficile l'alimentation des boîtiers ainsi que le découplage.

Pour une réalisation en petite série la gravure de circuits imprimés, tout au moins pour les unités de calcul, est la solution la plus rapide. La principale précaution à prendre consiste à réaliser des alimentations suffisamment dimensionnées, ou mieux à utiliser des barettes d'alimentation du type "bus-bar".

L'ensemble des cartes est placé dans un bac, les liaisons étant réalisées par un bus. Les cartes unité de calcul sont placées successivement, la dernière utilisée étant munie d'un cavalier reliant le signal de sélection à la ligne de test (cf. IV.2.2.3).

2 - PERFORMANCES

2.1. Vitesse de calcul

Le temps nécessaire au calcul de la commande est le produit de trois facteurs :

- le nombre d'itérations utiles pour résoudre le problème
- le nombre de microinstructions exécutées pendant une itération
- la durée d'une microinstruction

2.1.1. Nombre d'itérations

Il est difficile de donner le nombre d'itérations nécessaires pour résoudre un problème car il dépend de nombreux éléments :

- le choix du critère
- la précision demandée
- les contraintes
- l'état initial
- l'initialisation des trajectoires
- le paramètre de relaxation

Indiquons, en anticipant légèrement, que les exemples traités par la suite demanderont rarement plus d'une cinquantaine d'itérations pour être résolus.

2.1.2. Nombre de microinstructions par itération

Cette valeur est obtenue tout simplement en comptant les microinstructions exécutées sur le listing (annexe VI.1) entre les adresses 33 et 85 (hexadécimal). Nous obtenons :

$$N (69.I + 23.J + 60) + 10$$

avec

N : l'horizon

I : la dimension de l'état

J : la dimension de la commande

2.1.3. Durée du cycle microinstruction

La détermination de la durée du cycle microinstruction est plus complexe car il est indispensable de calculer le temps nécessaire pour que puissent se réaliser la totalité des opérations en tenant compte des éléments

technologiques employés. Nous faisons entrer en compte :

- les temps de propagation des signaux depuis le début de la période d'horloge (montée du signal)
- les temps de maintien des signaux avant la fin de la période d'horloge (montée suivante du signal).

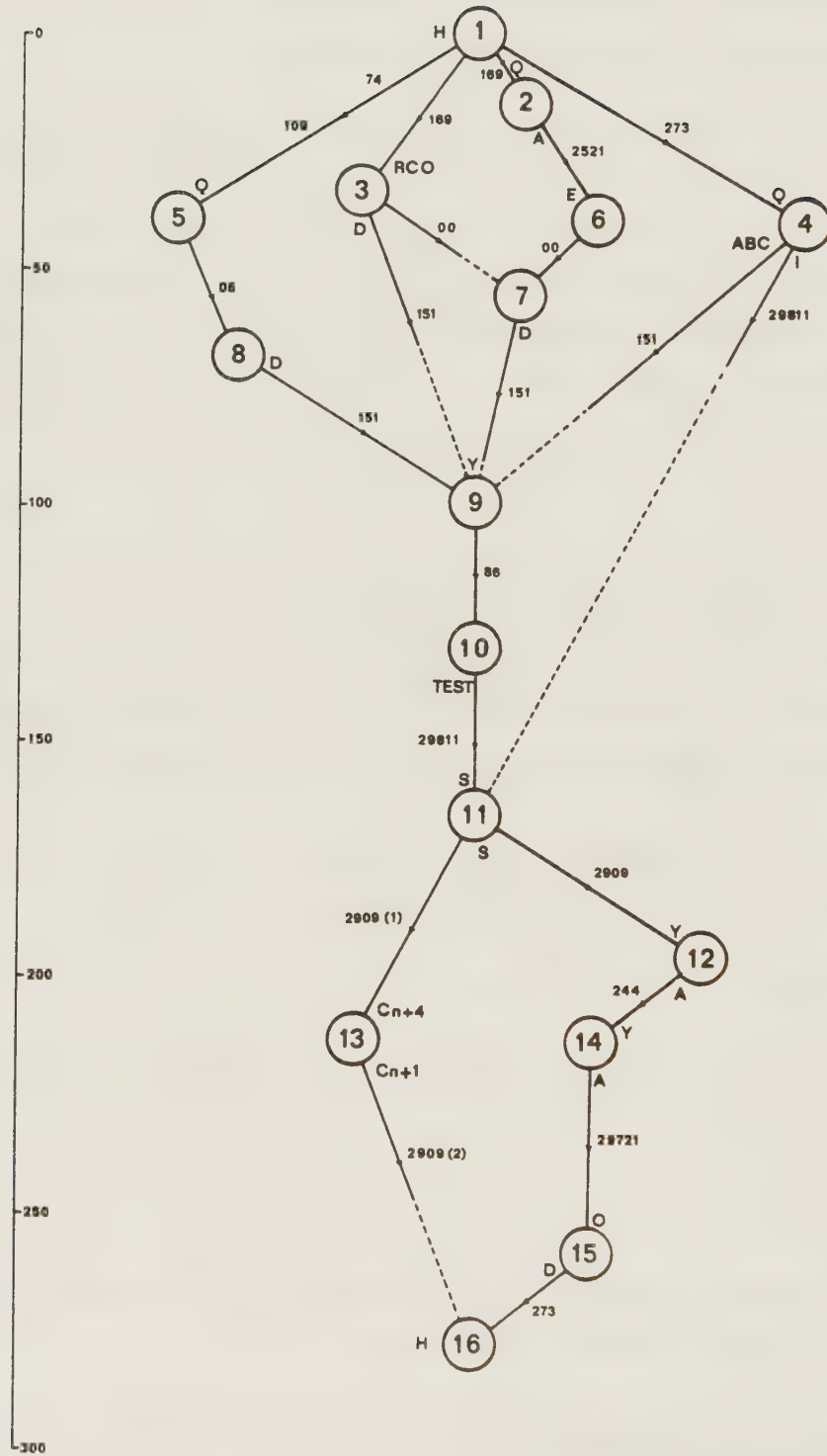
Pour faciliter cette opération et pour mettre en évidence les améliorations les plus rentables nous nous aidons d'un graphe. Chaque arc représente la propagation d'un signal dans un circuit intégré la longueur du trait continu étant proportionnelle au temps de propagation. Chaque sommet représente au contraire, des connexions entre des boîtiers. Nous portons sur les arcs la référence du circuit ainsi que les entrées et les sorties considérées alors que les sommets sont simplement repérés.

Nous donnons en exemple (figure 1) la portion relative à l'unité de commande que nous avons simplifiée pour la rendre plus lisible. Nous ne conservons que les chemins critiques (les plus longs) sauf lorsque le parcours dépend de la microinstruction exécutée ou lorsqu'il sera modifié lors de la réalisation définitive.

Etudions les chemins critiques entre 1 et 11 pour l'exécution de différentes microinstructions :

- instruction inconditionnelle : 1, 4, 11
- condition de rupture portant sur le compteur de boucle :
1, 4, 9, 10, 11
- condition portant sur le compteur de pas : 1, 2, 6, 7, 9
- condition portant sur une information en provenance d'une unité de calcul : 1, 5, 8, 9, 10, 11.

Entre les sommets 11 et 16 le chemin le plus long est 11, 12, 14, 15, 16 incluant les temps de propagation dans l'amplificateur de bus d'adresse le temps d'accès à la mémoire et le temps d'établissement à l'entrée du registre pipe-line. Si par la suite nous



utilisons une mémoire morte incluant le registre pipe-line nous remplaçons trois éléments par un seul et réduisons ainsi la longueur du chemin de moitié. Le chemin critique, pour la version définitive du calculateur sera alors 11, 13, 16.

La partie du graphe relative aux unités de traitement est donnée en annexe avec les temps nécessaires pour l'exécution des différentes fonctions. Il est ainsi possible de calculer la durée de chaque microinstruction en tenant compte des opérations réalisées. La durée d'une microinstruction est comprise entre 186 et 291 ns.

Les valeurs données ne sont bien sûr qu'une estimation de la réalité physique du système. En effet d'une part il n'est pas possible de prévoir exactement l'influence du câblage sur les performances, d'autre part les caractéristiques des circuits ne sont données que sous certaines conditions de température et de charge ne correspondant pas toujours aux conditions réelles. Nous estimons cependant que le choix d'une période d'horloge, fixe dans un premier temps, de 300 ns doit assurer un fonctionnement correct du calculateur en toutes circonstances.

2.1.4. Conclusion

Le temps total nécessaire à l'obtention de la loi de commande pour un problème de dimensions I et J dont le calcul nécessite K itérations est de :

$$K \left(N (69.I + 23.J + 60) + 10 \right) 300 \text{ ns}$$

Par exemple un problème d'ordre 4 à 2 commandes convergeant en 20 itérations pour un horizon de 31 demande environ 70 ms pour être résolu.

2.2. Performances réelles du calculateur

Les premières expériences sont menées sur le prototype muni des quatre unités de traitement déjà câblées. Nous résolvons des problèmes de dimension 4 pour le vecteur d'état et 2 pour la commande, l'horizon étant de 31. Nous noterons la vitesse de calcul d'une part, la précision obtenue d'autre part.

1.2.1. Vitesse

Une horloge de 300 ns est tout à fait satisfaisante et donne une marge suffisante de sécurité. Le temps nécessaire pour une itération est dans ces conditions 3,5 ms.

En utilisant le programme décrit dans le chapitre précédent (3.2) et implanté sur le calculateur hôte nous pouvons commander facilement l'exécution du calcul sur le calculateur vecteur pour obtenir les résultats et le temps nécessaire au calcul.

Les facteurs intervenant sur la durée de calcul sont fixés par le problème à résoudre (modèle, critère, horizon, précision ...) ou dépendent de l'architecture du système (période d'horloge) ou encore de l'écriture du microprogramme. Cependant nous pouvons choisir le paramètre de relaxation pour obtenir une vitesse de convergence maximale. Nous avons tracé (figure 2) la courbe représentant le temps de calcul en fonction du paramètre de relaxation pour le problème suivant :

soit le modèle :

$$x^{n+1} = A x^n + B u^n$$

avec :

$$A = \begin{bmatrix} 0,9 & 0,05 & 0,01 & 0,02 \\ 0,05 & 0,8 & 0,04 & 0,02 \\ 0,03 & 0,02 & 0,85 & 0,01 \\ 0,01 & 0,02 & 0,03 & 0,9 \end{bmatrix} \quad B = \begin{bmatrix} 0,01 & 0,03 \\ 0,02 & 0,01 \\ 0,04 & 0,01 \\ 0,02 & 0,03 \end{bmatrix}$$

le critère :

$$J = \frac{1}{2} \sum_{n=1}^N p (x^n)^T (x^n) + q (u^{n-1})^T u^{n-1}$$

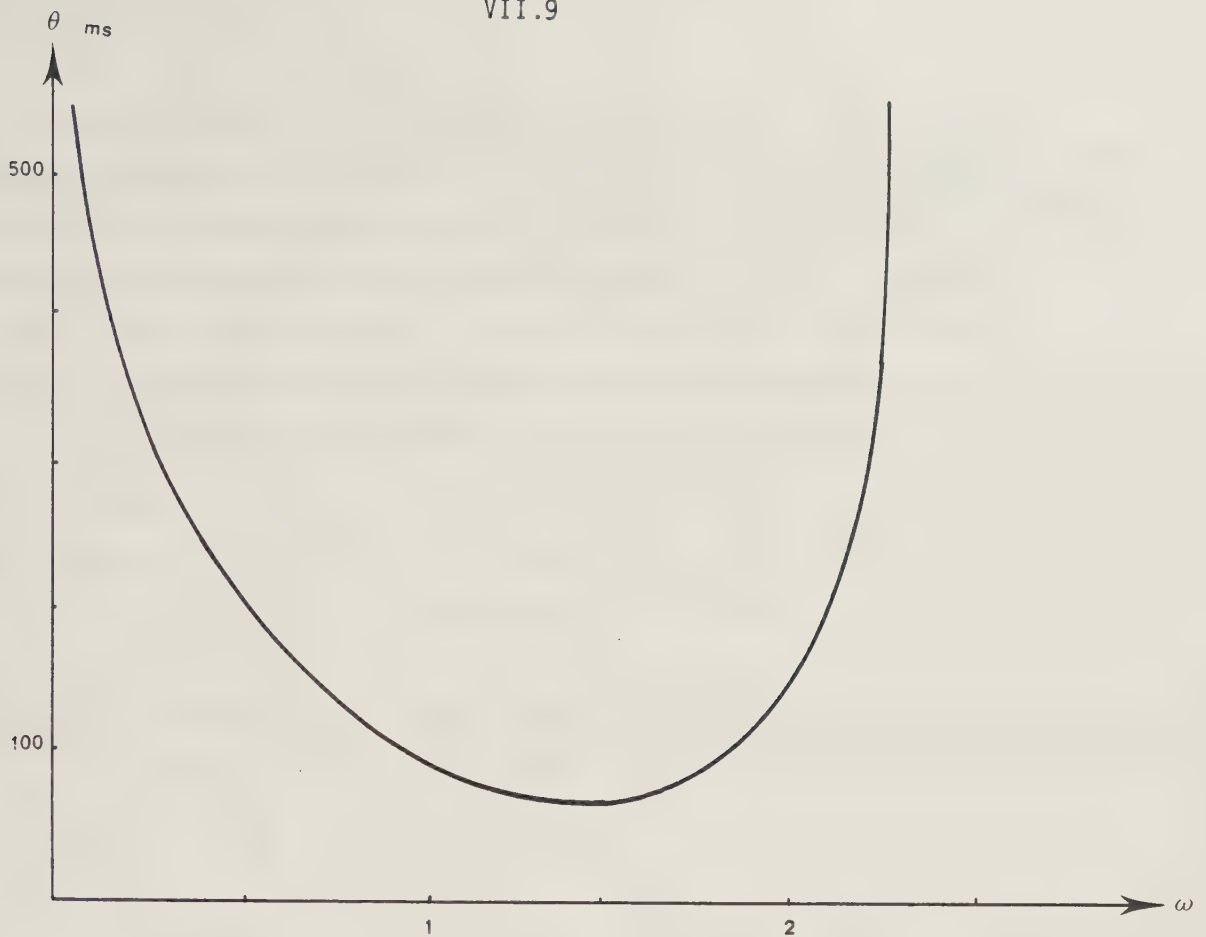


Figure 2 : Temps de calcul en fonction de ω

avec :

$$p = 0,2 \quad q = 0,01 \quad r = \frac{p}{q} = 20 \quad N = 31$$

l'état initial :

$$x^0 = \begin{bmatrix} 0,5 \\ 0,4 \\ 0,3 \\ 0,2 \end{bmatrix}$$

en prenant pour le test de convergence :

$$\delta = 10^{-4}$$

Nous reprenons ensuite ce problème et nous le traitons pour différentes valeurs du critère. Pour conserver une bonne précision nous modifions le cadrage des variables ainsi que le test de convergence (cf.2.2.2). Nous choisissons ensuite pour chaque problème le meilleur paramètre de relaxation. Les résultats sont résumés dans un tableau (figure 3) où les valeurs de p et q sont données avant cadrage. Nous remarquons immédiatement que la vitesse de convergence varie beaucoup avec le critère. Les performances du calculateur assurent cependant un temps de calcul limité ; la valeur de 250 ms étant un maximum pour la plupart des problèmes que nous avons traités sur le calculateur.

N°	p	q	r	δ	ω	θ ms
1	.2	.1	2	10^{-3}	.5	< 38
1	.2	.02	10	10^{-3}	.3	49
3	.2	.02	10	10^{-4}	.3	78
4	.2	.01	20	10^{-4}	.15	92
5	.4	.01	40	10^{-4}	.075	165

Figure 3 Temps de calcul pour différents critères

2.2.2. Précision

Afin d'étudier la précision nous comparons les résultats des calculs effectués sur le calculateur vecteur avec les mêmes résultats obtenus en virgule flottante sur le calculateur hôte. Afin que le test de convergence ne vienne pas fausser l'expérience nous le choisissons aussi petit que possible ce qui accroît le temps de calcul et risque même d'en empêcher l'arrêt. L'erreur de calcul due à des arrondis est aléatoire et peu significative ; aussi pour ces expériences nous procéderons à des troncatures donnant ainsi une vision pessimiste des performances du calculateur.

Nous avons remarqué principalement que l'erreur de calcul variait avec la vitesse de convergence. Elle est généralement inférieure à 10^{-4} pour les problèmes convergeant rapidement (20 itérations) alors qu'elle se rapproche de 5.10^{-4} pour les problèmes convergeant plus lentement (50 itérations).

Nous pouvons rapprocher cette expérimentation des résultats théoriques obtenus au chapitre IV (1.4.3).

La relation :

$$| \tilde{U}_{p+1} - U | \leq \epsilon + C | U_p - U | \quad C < 1$$

montre que la convergence est d'autant plus rapide que C est petit. L'erreur est donnée par :

$$| \tilde{U}_{p+1} - U | \leq \frac{1}{1-C} \epsilon + \frac{C}{1-C} \delta$$

et nous constatons qu'elle devient plus importante quand C se rapproche de 1 c'est-à-dire quand la convergence est plus lente.

Si nous arrondissons les variables au lieu d'effectuer une simple troncature nous obtenons alors une erreur inférieure à 10^{-4} pour la plupart des problèmes.

2.3. Performances du calculateur complet

Nous pouvons extrapoler les résultats obtenus pour évaluer les performances d'un calculateur de dimension 16.

Pour des problèmes équivalents (demandant le même nombre d'itérations) le temps de calcul est proportionnel à la dimension des vecteurs. Un problème d'ordre 16 avec une commande de dimension 8 peut être résolu dans un temps inférieur à une seconde.

La précision du calcul est, pour des problèmes équivalents (mêmes normes des matrices), proportionnelle à la racine carrée de la dimension des vecteurs. Nous obtenons dans ces conditions une précision de l'ordre de 2.10^{-4} .

Ces valeurs sont approximatives et dépendent du problème à résoudre mais donnent cependant une idée des performances du calculateur. Des résultats plus précis pourront être obtenus pour chaque problème particulier en utilisant le programme de simulation, ou le calculateur vecteur si nous disposons de suffisamment d'unités de traitement.

2.4. Amélioration des performances

Nous recherchons des solutions permettant d'augmenter la vitesse de calcul sans modifier l'architecture du calculateur.

Le temps de cycle microinstruction peut être réduit en remplaçant les composants par des équivalents plus rapides. Citons par exemple le circuit Am 2901 C [28] pour l'unité de traitement ainsi que la série TTL FAST [29] pour toutes les fonctions simples.

L'unité de commande définitive sera plus performante après remplacement de l'ensemble buffer d'adresse, mémoire de commande et registre pipe-line par un seul élément (Am 27 S 20 [29]). Il est également possible de remplacer les séquenceurs 2909 et les circuits associés 29811 et 169 par un séquenceur 2910 plus rapide. L'horloge programmable non utilisée lors des premiers essais, accélèrera encore le déroulement des microprogrammes (gain de 20 %) au prix d'une légère complication de la programmation.

Ces diverses améliorations divisent par deux le temps de calcul théorique. Certaines accroissent le coût du calculateur (technologie plus rapide) mais les autres s'imposent pour une réalisation définitive. Si nous désirons réaliser un calculateur peu coûteux nous ne retiendrons que les dernières, le cycle d'horloge étant alors d'environ 200 ns.

La moitié du temps de calcul est consacré à l'exécution de la multiplication microprogrammée. L'utilisation de multiplieurs rapides est la modification la plus efficace pour accroître la vitesse de calcul. Chaque multiplieur est placé (figure 4) de manière à travailler en parallèle avec l'unité arithmétique. Cette disposition ainsi que la présence de registres à l'entrée et à la sortie du multiplieur permet d'effectuer en deux cycles une multiplication suivie d'une addition (ces opérations incluant le chargement des registres X et Y et la récupération d'une partie du mot de poids faible du produit XY).

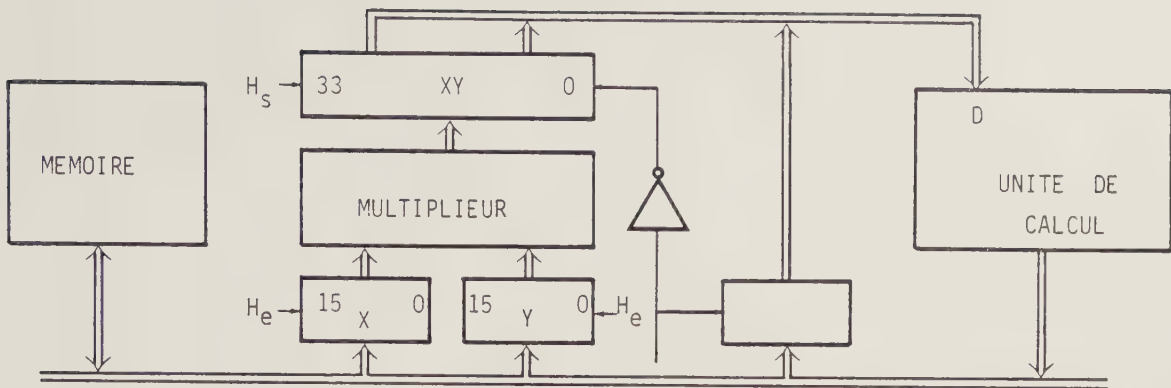


Figure 4 : Unité de traitement avec multiplieur

3 - CONDUITE D'UN PROCESSUS SIMULE

3.1. Conduite centralisée

Afin de tester dans des conditions proches de l'utilisation réelles le calculateur vecteur nous lui avons fait conduire un processus simulé sur le calculateur hôte.

3.1.1. Commande d'un processus parfait

Nous désirons conduire de façon optimale le processus d'un état initial donné à l'état 0 en un temps infini en utilisant la méthode de conduite en boucle fermée par intermittence décrite au chapitre I (5.2.1). Nous supposons qu'il n'existe pas de perturbations et que le modèle est exact. Le calculateur hôte détermine, à chaque instant, l'état du processus, l'envoie au calculateur vecteur puis reçoit la commande qu'il utilisera pour calculer la valeur suivante de l'état. Pour réduire le temps de calcul nous conservons, pour initialiser les trajectoires, les résultats précédents.

Nous reprenons l'exemple du paragraphe 2.2.1. et nous conduisons le processus pour les différentes valeurs de r . Nous avons tracé l'évolution des variables d'état et des commandes pour $r = 2$ (figure 5) et $r = 40$ (figure 6).

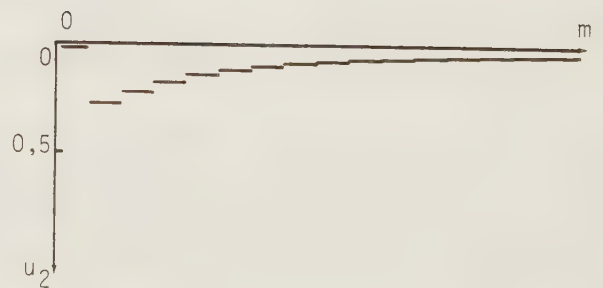
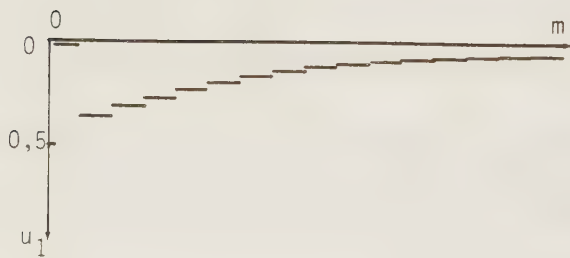
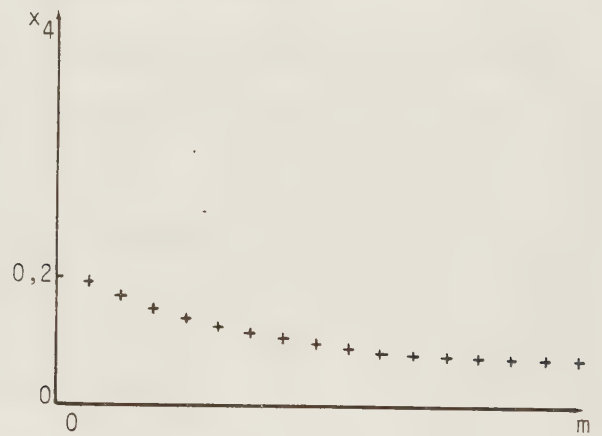
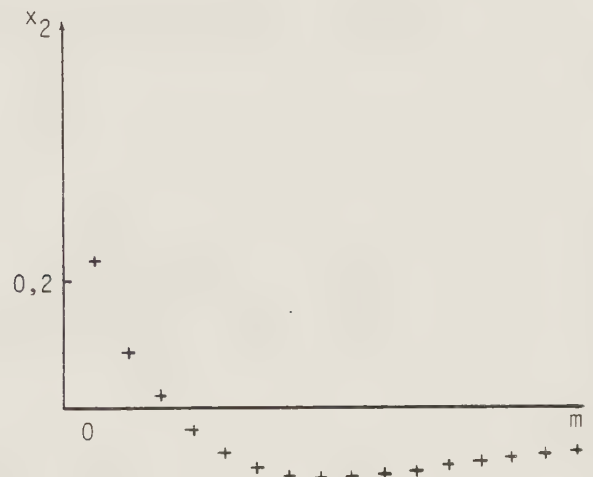
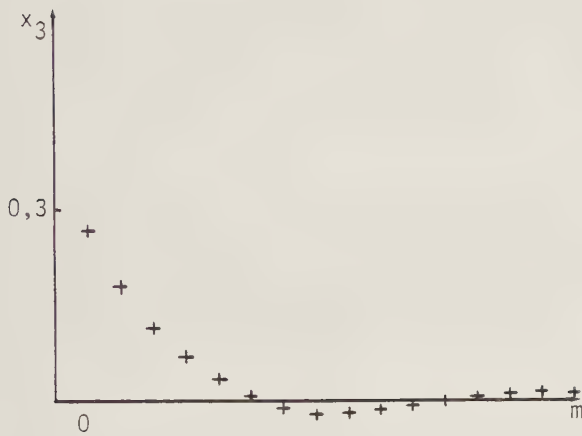
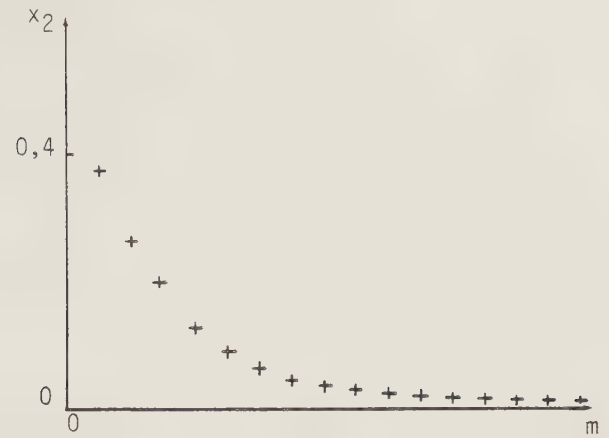
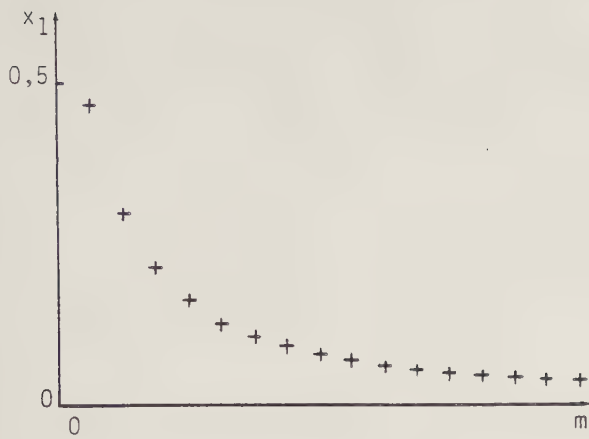
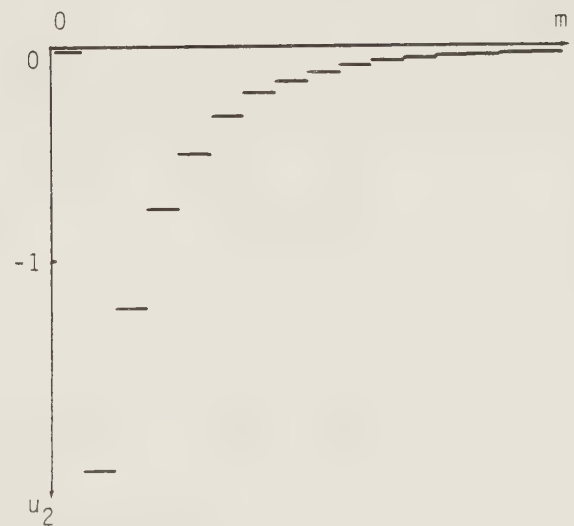
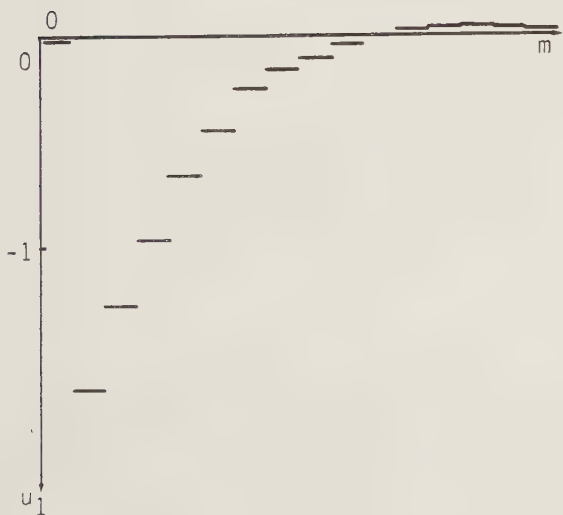


Figure 5 : Conduite en boucle fermée par intermittence du processus pour $r = 2$

VII.15



Evolution de l'état



Evolution de la commande

Figure 6 : Conduite en boucle fermée par intermittence du processus pour $r = -$

Dans le premier cas le temps de calcul est toujours très petit alors que dans le deuxième il est relativement important surtout en début de trajectoire. Le premier calcul effectué à partir d'une trajectoire $\{u^n\}^0$ nulle est particulièrement long (figure 7

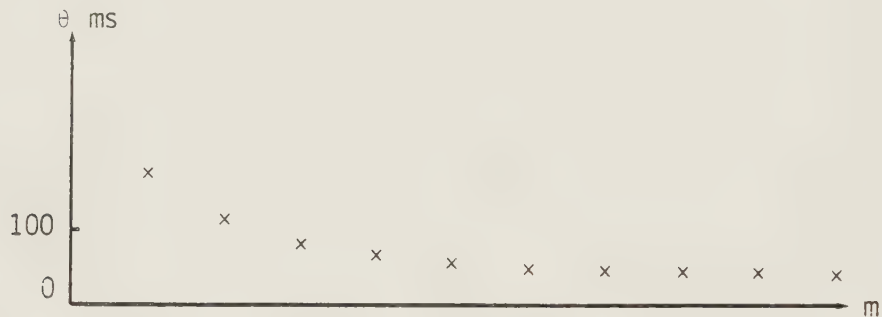


Figure 7 : Evolution de la durée du calcul en fonction du temps

Cet inconvénient peut être supprimé en arrêtant le calcul au bout d'un temps fixé (par exemple 100 ms dans ce cas) au prix d'une légère perte de précision sans effet notable sur l'évolution du processus.

La période d'échantillonnage peut être réduite (ici 100 ms) ce qui permet de conduire des processus très rapides même si le calcul de la commande est intrinsèquement long (une cinquantaine d'itérations).

3.1.2. Régulation d'un processus perturbé

Nous supposons maintenant que le processus est régi par l'équation :

$$x^{m+1} = A x^m + B u^m + \epsilon^m$$

dans laquelle ϵ^m représente les perturbations et l'erreur de modélisation. Nous reprenons le problème déjà traité au paragraphe précédent et nous observons l'évolution au voisinage de l'état 0. Nous prenons comme perturbation ϵ^m égal à un membre aléatoire compris entre 0 et 0,05.

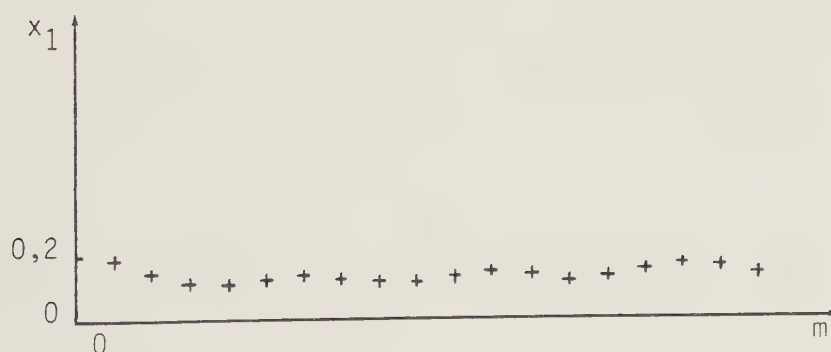
Nous constatons que le processus diverge en boucle ouverte (figure 8) à cause de l'importance et du signe constant de la perturbation ε^m . La conduite du processus en boucle fermée par intermittence maintient, au contraire, le système du voisinage du point de régulation. Nous avons tracé en boucle ouverte et pour $r = 40$ l'évolution de l'état x_1 , les trajectoires des autres variables étant similaires. Le calcul de la commande peut être effectué en moins de 50 ms tout en conservant une bonne précision.

3.1.3. Commande d'un processus instable

Nous choisissons maintenant un problème plus complexe :



Boucle ouverte



Régulation ($r = 40$)

Figure 8 : Processus perturbé

la conduite d'un processus instable en boucle ouverte.

Soit le modèle :

$$A = \begin{bmatrix} 0,95 & 0,025 & 0,03 & 0,04 \\ 0,05 & 0,9 & 0,04 & 0,05 \\ 0,01 & 0,005 & 1 & 0,3 \\ 0,06 & 0,002 & 0,01 & 0,95 \end{bmatrix} \quad B = \begin{bmatrix} 0,1 & 0,3 \\ 0,3 & 0,05 \\ 0,05 & 0,3 \\ 0,15 & 0,25 \end{bmatrix}$$

le critère :

$$p = 0,4 \quad q = 1 \quad N = 31$$

l'état initial :

$$x^0 = \begin{bmatrix} 0,1 \\ 0,1 \\ 0,1 \\ 0,1 \end{bmatrix}$$

et le test :

$$\delta = 10^{-4}$$

Nous prenons pour perturbation ϵ^m un nombre aléatoire compris entre $-0,025$ et $+0,025$. Le tracé des variables d'état effectué pour les premiers instants (figure 9) se poursuit par de faibles oscillations au gré des perturbations. Cette expérience est beaucoup plus difficile pour le calculateur vecteur car d'une part le calcul converge très lentement et d'autre part les variations de l'état sont importantes d'une période à l'autre ne permettant pas ainsi une bonne initialisation des trajectoires. Le temps de calcul ne peut être pris inférieur à 300 ms en régime permanent et atteint même 1 s pour le calcul initial (état x^1 et trajectoires à 0).

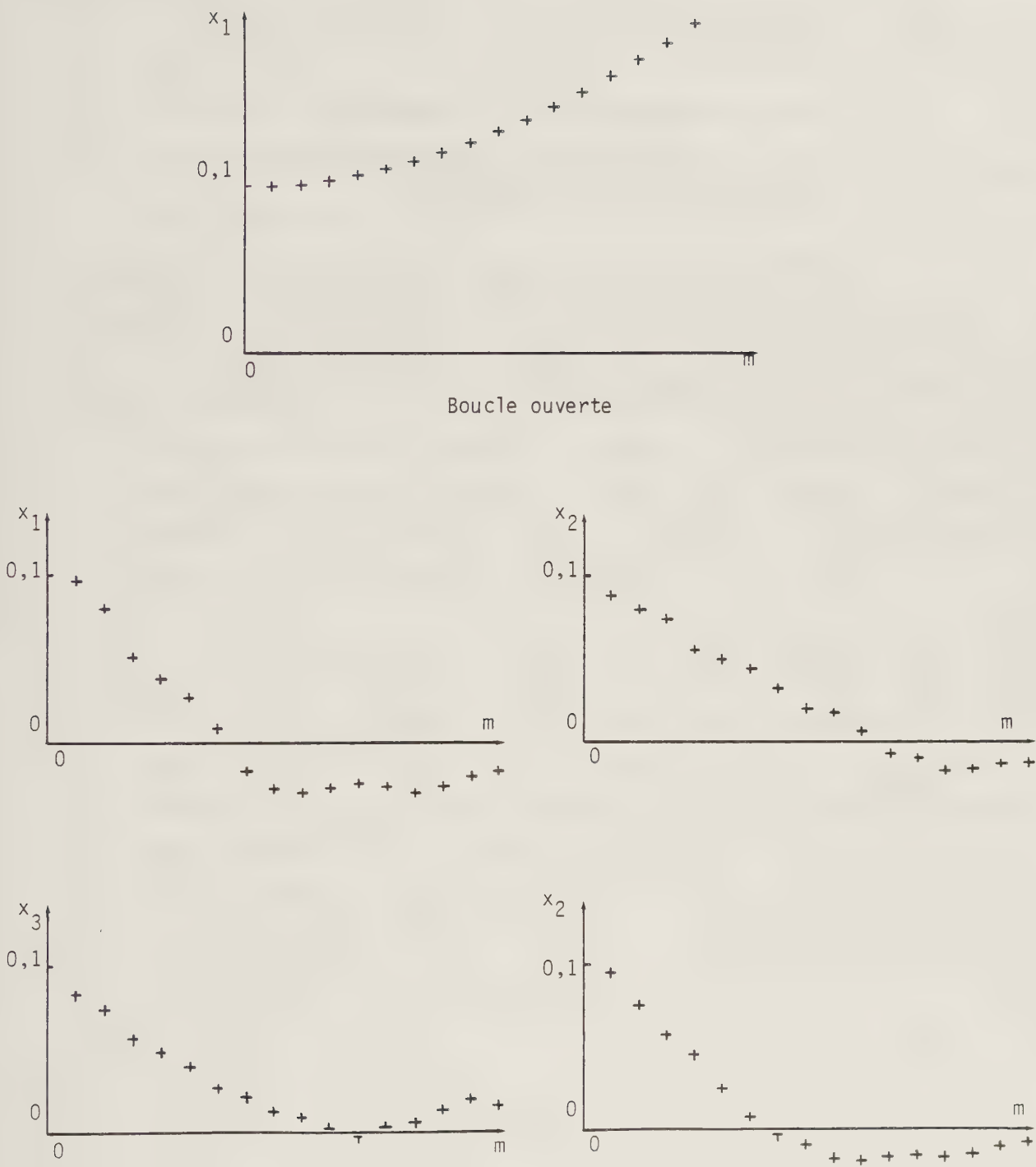


Figure 9 : Conduite d'un processus instable et perturbé

3.2. Simulation de conduite décentralisée

Nous avons simulé la conduite décentralisée de processus inter-connectés par un système informatique distribué . Nous analysons la précision obtenue ainsi que le temps nécessaire aux calculs en considérant trois cas :

- conduite décentralisée sans coordination
- conduite décentralisée par une méthode sous-optimale
- conduite optimale (pour comparaison).

Nous éliminons les incertitudes dues aux calculs en effectuant ceux-ci avec une bonne précision. Nous déterminons sur le même calculateur l'évolution des processus et les commandes calculées de la manière choisie. Prenons un exemple simple comprenant deux processus ayant pour modèles :

$$A_{11} = \begin{bmatrix} 0,9 & 0,05 \\ 0,05 & 0,8 \end{bmatrix} \quad B_{11} = \begin{bmatrix} 0,1 \\ 0,2 \end{bmatrix}$$

$$A_{22} = \begin{bmatrix} 0,85 & 0,01 \\ 0,03 & 0,9 \end{bmatrix} \quad B_{22} = \begin{bmatrix} 0,1 \\ 0,3 \end{bmatrix}$$

couplés par les matrices :

$$A_{12} = \begin{bmatrix} 0,01 & 0,02 \\ 0,04 & 0,02 \end{bmatrix} \quad A_{21} = \begin{bmatrix} 0,03 & 0,02 \\ 0,01 & 0,02 \end{bmatrix}$$

en partant de l'état :

$$x_1^o = \begin{bmatrix} 0,3 \\ 0,2 \end{bmatrix} \quad x_2^o = \begin{bmatrix} 0,3 \\ 0,2 \end{bmatrix}$$

avec le critère :

$$p = 0,4$$

$$q = 1$$

$$N = 31$$

et le test :

$$\delta = 10^{-4}$$

Nous avons tracé (figure 10) l'écart Δx_1 entre la trajectoire x_1 obtenue par la méthode optimale et celles obtenues par la méthode sous-optimale et sans coordination.

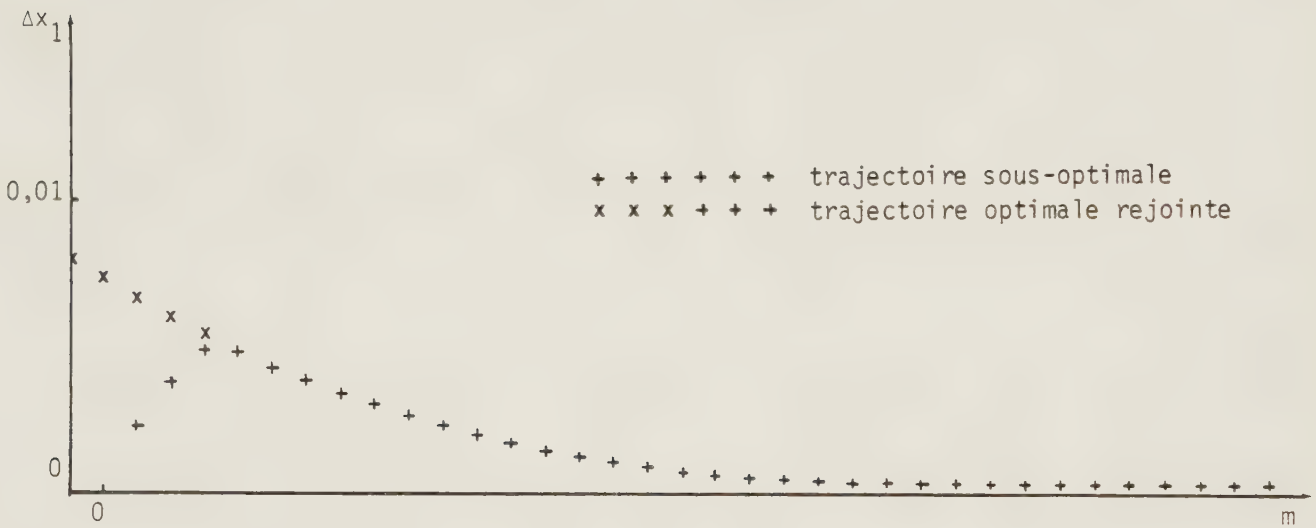
Si l'on ne tient pas compte des trajectoires de couplage l'écart devient vite assez important pour décroître lentement par la suite. Au contraire la méthode sous-optimale décrite au chapitre II (3.2.2.1) donne une erreur limitée et décroissant rapidement dès l'instant 5. A partir de là le système suit exactement (écart inférieur à 10^{-4}) une trajectoire optimale. L'erreur obtenue au début de la trajectoire provient de la mauvaise initialisation à l'instant 0. Les trajectoires de couplage n'ayant pas encore été échangées le premier calcul est donc effectué sans coordination et conduit à une commande fortement erronée entre les instants 1 et 2. Cet inconvénient disparaît dans le cas de la régulation d'un processus.

L'utilisation de la méthode décentralisée avec coordination divisée, dans ce cas, par deux le temps de calcul au niveau (du et) des calculateurs vecteurs. Pour que cet avantage se ressente sur le temps global du traitement il est nécessaire que le transfert des variables et le calcul des trajectoires de couplage soit suffisamment rapide (les calculateurs vecteurs calculent ici les commandes en moins de 30 ms). Ce problème peut être résolu par l'utilisation d'une architecture multiprocesseur pour le calculateur hôte (cf. VI.2.4.1) afin de mener en parallèle la conduite du processus, les échanges et le calcul des termes de couplage.

VII.22



Sans coordination



Avec coordination

Figure 10 : Conduite décentralisée

CONCLUSION

Le calculateur vecteur est un outil de calcul très performant permettant la conduite optimale de processus rapides. La précision obtenue est suffisante pour la commande de processus industriels en raisons d'incertitudes sur les mesures et la modélisation et du choix plus ou moins arbitraire du critère. Nous remarquons au cours des expériences de conduite d'une simulation de processus que le temps de calcul est très faible dans le cas de la régulation lorsque l'état varie peu entre deux instants consécutifs.

Nous notons par ailleurs que les erreurs de calcul ont peu d'influence sur l'évolution du processus commandé.

La conduite décentralisée de processus interconnectés par la méthode sous-optimale semble donner d'excellents résultats. Exceptés les premiers instants nécessaires à l'initialisation des trajectoires de couplage le système suit avec exactitude une trajectoire optimale.

ANNEXE VII . 1

TEMPS D' EXECUTION

Unité de commande

Instruction inconditionnelle :

1, 4, 11, 12, 14, 15, 16 186 ns

Condition portant sur le compteur de boucles :

1, 4, 9, 10, 11, ... 259 ns

Condition portant sur le compteur de pas :

1, 2, 6, 7, 9, 267 ns

Condition portant sur une information en provenance
d'une unité de traitement

1, 5, 8, 9, 280 ns

Unité de traitement

Opération avec source mémoire ou entrée

Lecture :

Lecture en mémoire et transfert entre unités :

1, 4, 19, 21, 29, 36, 37 188 ns

Entrée de données et transfert aux unités :

1, 4, 20, 23, 26, 36, 37 141 ns

Lecture en mémoire locale :

1, 4, 19, 21, 37 128 ns

Calcul :

Pas d'opération arithmétique (chargement d'un registre) :

37,16 60 ns

Décalage numérique sans opération arithmétique :

37, 52, 53, 16 95 ns

Opération arithmétique :

37, 45, 44, 53, 16 103 ns

Opérations avec destination mémoire ou sortie

Calcul :

Sortie directe d'un registre :

1, 4, 20, 25, 39 121 ns

Pas d'opération arithmétique :

1, 4, 20, 39 131 ns

Opération arithmétique :

1, 4, 20, 30, 31, 39 149 ns

Opération arithmétique suivant une multiplication
ou le calcul de la valeur absolue

1, 4, 20, 24, 30, 31, 39 167 ns

Ecriture :

Ecriture en mémoire :

39, 38, 16 (avec $t_H = 100$ ns)Sortie :

39, 46, 16 30 ns

Transfert entre registres

1, 4, 20, 16 149 ns

Opération arithmétique

Sans décalage

1, 4, 20, 33, 32, 16 177 ns

Sans décalage suivant une multiplication ou le calcul de la valeur absolue

1, 4, 20, 26, 33, 32, 16 195 ns

Avec décalage

1, 4, 20, 33, 32, 40, 47, 16 198 ns

Après décalage suivant une multiplication ou le calcul de la valeur absolue

1, 4, 20, 26, 33, 32, 40, 47, 16 216 ns

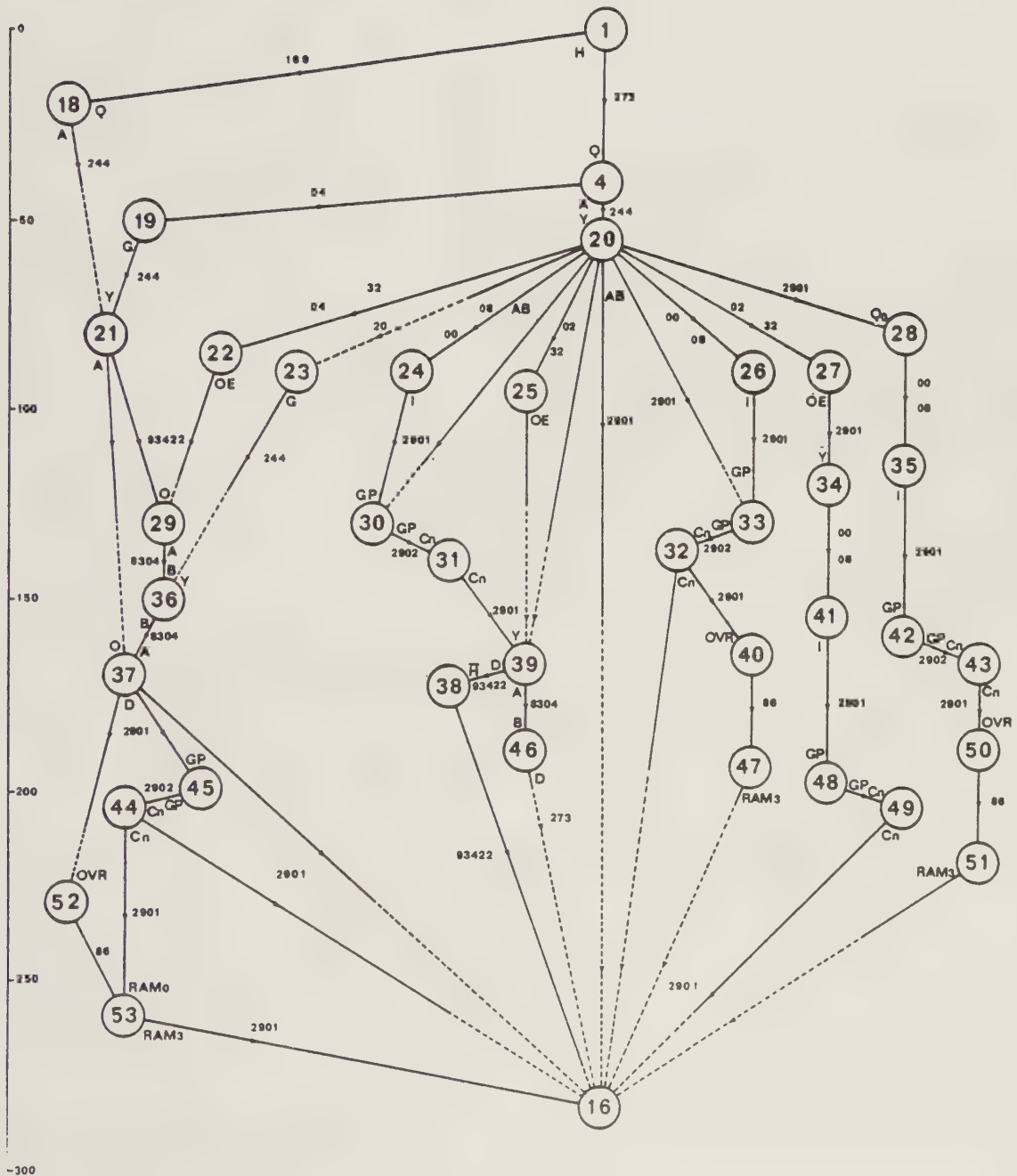
Multiplication

1, 4, 20, 28, 35, 42, 43, 50, 51, 16 241 ns

Calcul de la valeur absolue

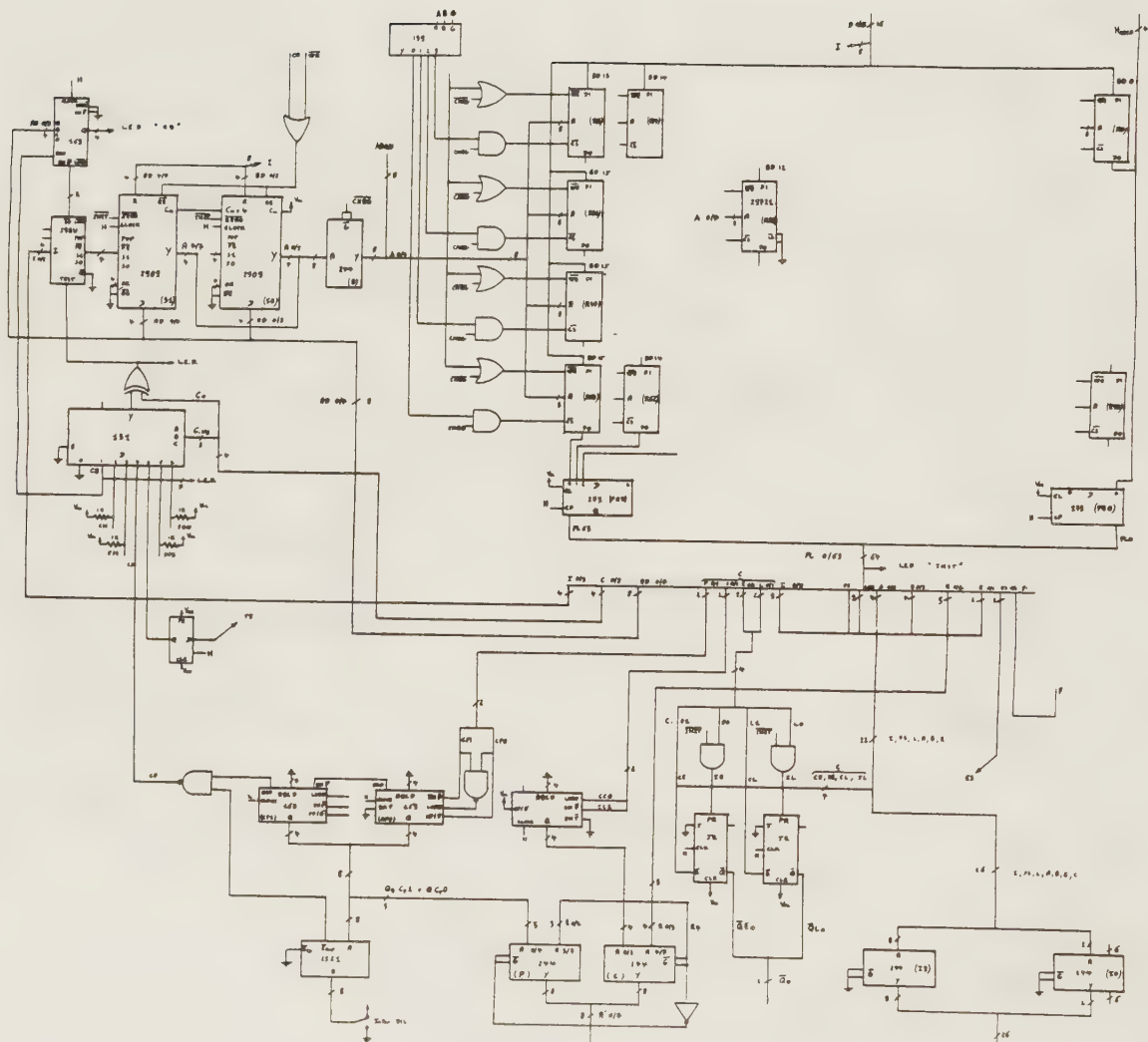
1, 4, 20, 27, 34, 41, 48, 49, 16

260 ns

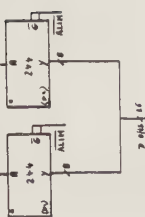
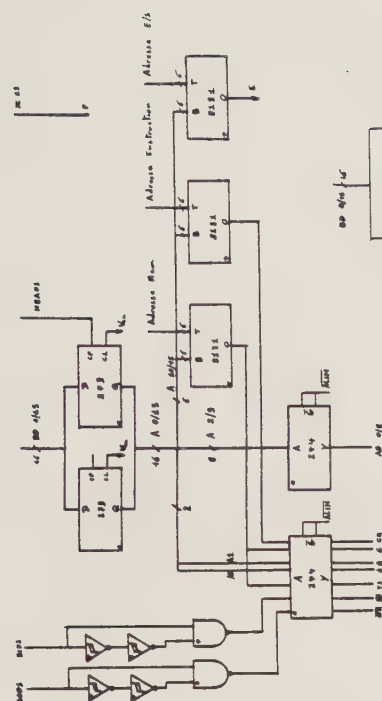


ANNEXE VII. 2

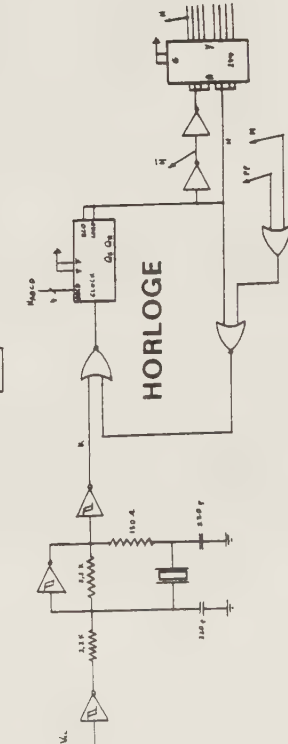
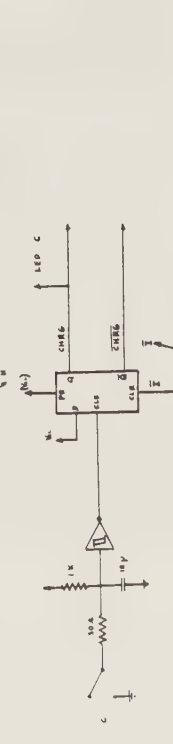
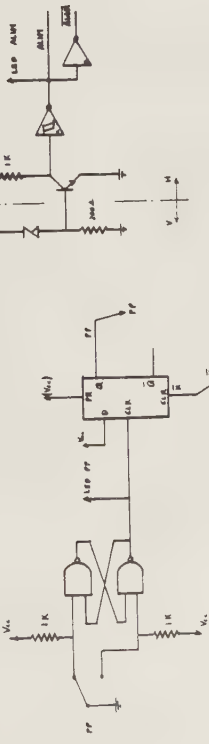
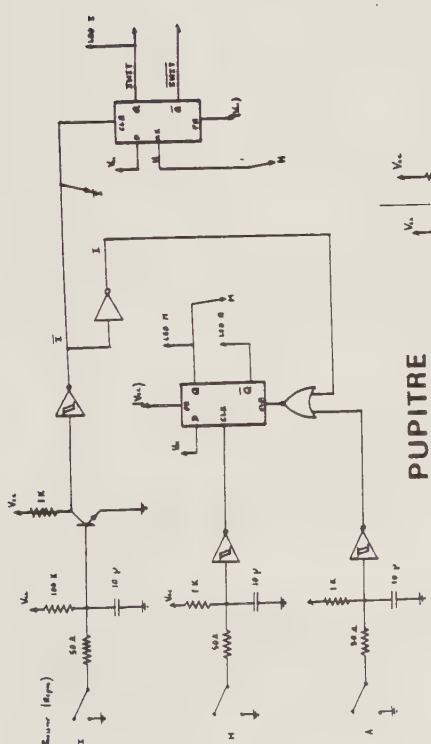
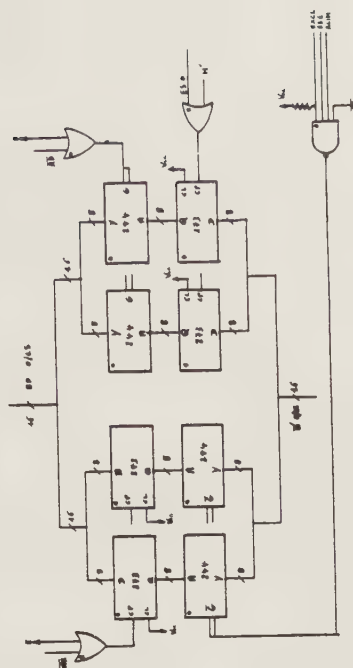
PLANS DU CALCULATEUR VECTEUR

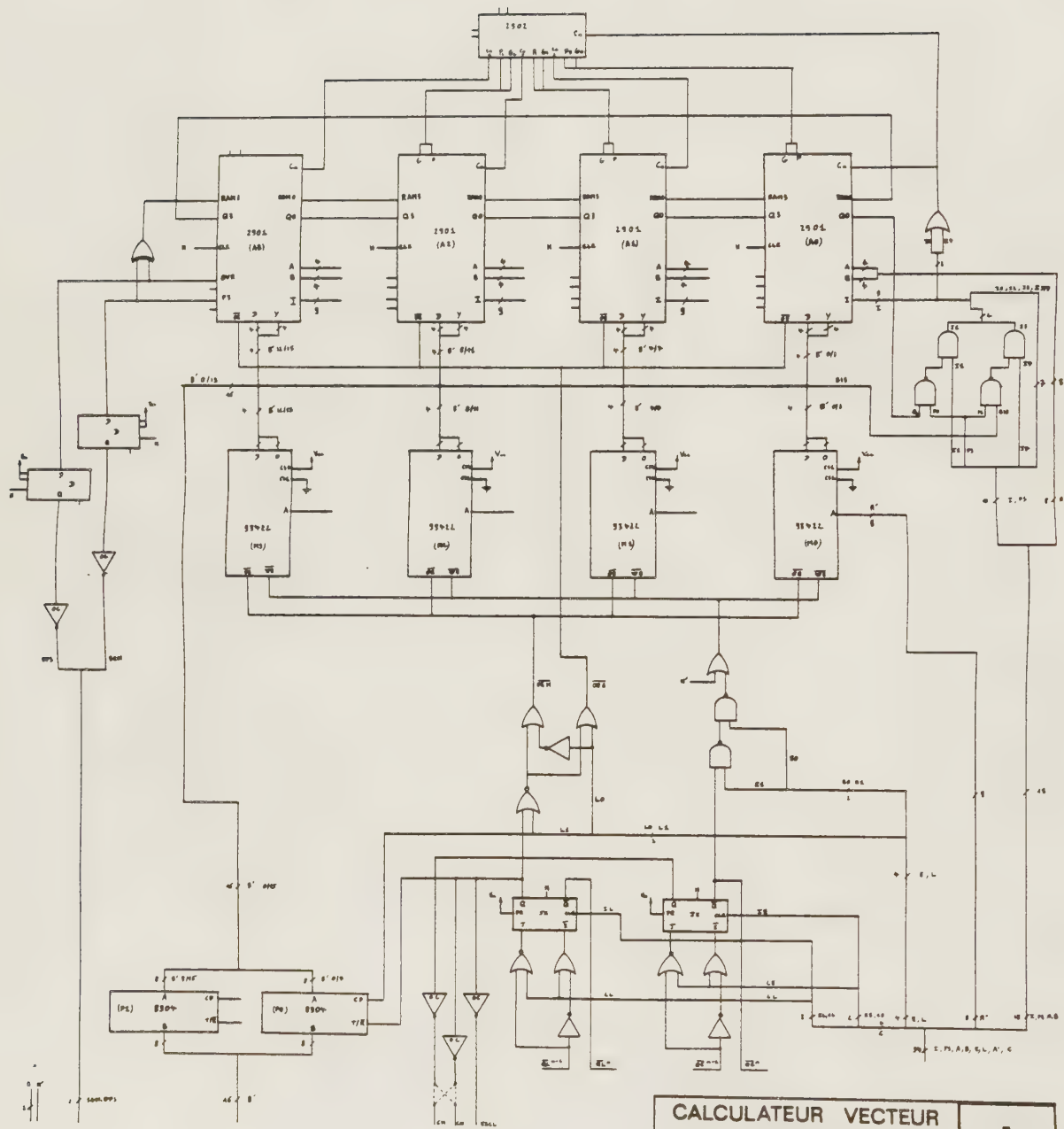


CALCULATEUR VECTEUR COMMANDE	LAB
--	------------



INTERFACE





CALCULATEUR VECTEUR TRAITEMENT	LAB
3344	

ANNEXE VII . 3

UTILISATION DU CALCULATEUR VECTEUR

I - Le calculateur vecteur

I.1. Le pupitre de commande

Le pupitre de commande permet le chargement du microprogramme dans la mémoire vive et le lancement de l'exécution.

INIT : initialise le calculateur et en particulier le compteur ordinal

MARCHE : lance l'exécution

ARRET : interrompt l'exécution

PAS/PAS: permet l'exécution pas à pas

CHARG : la mémoire de microprogramme est chargée depuis le calculateur hôte.

Pour exécuter un microprogramme il est nécessaire de le charger depuis le calculateur hôte à l'aide du chargeur (CHR) le calculateur vecteur étant en mode "CHARG". Le déroulement est ensuite lancé par "INIT" puis "MARCHE".

I.2. Le pupitre de visualisation

Le pupitre de visualisation permet de contrôler l'exécution du microprogramme principalement lors d'une exécution en mode pas à pas.

Adresse : adresse de la prochaine microinstruction

Instruction : instruction en cours d'exécution

Test : conditions

- 7 : SGN ; signe ; un résultat est négatif
 - 6 : DPS ; dépassement ; une unité arithmétique déborde
 - 5 : TS ; condition externe
 - 4 : CP ; compteur de pas ; (=0) + (= maxi)
 - 3 : CM ; registre à décalage
 - 2 : CN ; registre à décalage
 - 1 : CB ; compteur de boucle
 - 0 : TEST ; sortie de la logique de sélection
- } fin de vecteur

Compteur de pas : 5 bits sont utilisés ; 0 ... 4 ; 0 → maxi < 32

Compteur de composante : 0 → 15

Compteur de boucle : Valeur chargée → 0

Alimentation : contrôle tension alimentation $V_{CC} > 4,5 \text{ V}$
contrôle isolement de l'U.D.S.

I.3. Echanges avec le calculateur hôte

a - Chargement de la mémoire de commande

Il s'effectue comme le chargement d'une zone mémoire du calculateur hôte comprise entre les adresses 7000 et 73 FF. Chaque mot de 64 bits étant composé de 4 mots de 16 bits avec les poids forts situés à l'adresse la plus faible.

b - Chargement de l'instruction

Le registre instruction est situé à l'adresse 7400 et se charge comme un mot mémoire, les 8 bits de poids faible étant les seuls utilisés.

c - Echange de données

Le coupleur d'entrée sortie de 16 bits est adressé en lecture et en écriture à l'emplacement 7800. L'accès au coupleur provoque une condition de saut permettant la prise en compte de l'information par le calculateur vecteur.

II - Programmation du calculateur vecteur

Le microassembleur permet de créer à partir d'un fichier source écrit en langage symbolique un fichier objet binaire destiné à être chargé sur le calculateur vecteur.

Fichier source

Le fichier source est créé à l'aide de l'éditeur du texte. Les règles à respecter sont les suivantes :

- Chaque ligne n représente une microinstruction qui sera chargée à l'adresse n-1 dans la mémoire de microprogramme.
- Dans une ligne les différentes parties de la microinstruction sont séparées par au moins un espace.
- Les différents éléments de la microinstruction sont donnés dans l'ordre suivant :

Commande du séquenceur

Sélection de la condition

Compteur de pas

Compteur de coefficients

Adresse de saut

Compteur écriture

Compteur lecture

Destination unité arithmétique et logique

Opération

Source

Sélection d'une fonction spéciale

Sélection de la source (lecture)

Sélection du compteur

Commande de l'écriture en mémoire

Contrôle des entrées-sorties

Adresse du registre A

Adresse du registre B

Adresse (poids forts) mémoire

Flag

(Commande horloge)*

(Alarme)*

Commentaire : (facultatif)

- La longueur totale de la ligne ne doit pas excéder 99 caractères.

* non utilisé dans la version A

Fichier objet

Le fichier objet est un fichier de type symbolique contenant la représentation sous forme hexadécimale du code objet binaire. Cette manière de procéder permet de modifier le programme sans qu'il soit nécessaire d'effectuer un nouvel assemblage.

Ce fichier possède un nom du type (sauf indication contraire) :

NOM . PRG

Listing

Le listing est imprimé pendant l'assemblage. Pour chaque ligne sont indiqués successivement :

- l'adresse d'implantation
- le code hexadécimal
- la ligne source

La longueur d'une ligne est de 125 caractères. Les erreurs rencontrées sont signalées et repérées dans la ligne par le caractère .
L'assemblage peut être arrêté à tout instant en frappant une touche quelconque du clavier.

Le chargeur

Le programme chargeur permet par la suite de transférer le code objet sur le calculateur vecteur.

Modification de la microinstruction

Pour permettre des modifications aisées des différents éléments de la microinstruction l'assembleur comprend deux parties :

- l'assembleur général utilisable pour n'importe quelle architecture du calculateur
- une table donnant les codes instructions, la position des différentes valeurs et tous les éléments précisant la configuration du système.

Cette table peut être facilement écrite par l'utilisateur de l'assembleur en langage symbolique, assemblée puis jointe à l'assembleur général afin de former l'assembleur particulier à l'utilisation envisagée.

III - Logiciel implanté sur le système

Nous avons utilisé le micro-programme de calcul de la commande optimale décrit plus loin (PROGRAMME DE CALCUL) conjointement avec un programme écrit en langage BASIC destiné à l'exploitation des résultats. La marche à suivre pour exploiter ces programmes est la suivante :

Charger le micro-programme de calcul (CALCUL. PRG) sur le calculateur vecteur puis le lancer. Charger le BASIC sur le calculateur hôte puis exécuter le programme d'exploitation (VECTEUR. BAS). Ce programme conversationnel ne pose pas de difficultés d'emploi. Il permet :

- l'acquisition des paramètres depuis le clavier
- la modification de certains paramètres
- l'écriture des paramètres sur disque
- la lecture des paramètres sur disque
- l'impression des paramètres
- le calcul de la commande sur le calculateur vecteur et l'impression des résultats et du temps de calcul
- l'appel des programmes de calcul sur le système de développement et de simulation

CONCLUSION GÉNÉRALE

Après une rapide présentation du problème général de commande optimale et le choix d'une méthode de résolution nous nous sommes intéressés plus particulièrement à la conduite de processus à modèle linéaire, critère quadratique et contraintes sur la commande. Nous avons envisagé pour la conduite de systèmes complexes composé de processus interconnectés la décomposition des calculs puis la décentralisation du système de commande. Nous avons ensuite examiné les architectures informatiques parallèles et distribuées susceptibles d'être utilisées pour la mise en oeuvre des méthodes proposées.

Nous nous sommes orientés vers la conception d'un élément particulier du système : un calculateur vecteur destiné à effectuer rapidement les calculs itératifs vectoriels nécessaires à la détermination de la commande. Ce calculateur a été réalisé puis utilisé pour conduire quelques processus simulés. Les performances obtenues lui permettent de rivaliser avec un calculateur hybride et autorisent la conduite en ligne de nombreux processus.

Les possibilités du calculateur vecteur ont encore été très peu exploitées et nous envisageons dans un proche avenir son utilisation pour la résolution de problèmes non linéaires ainsi que son emploi pour la conduite d'un procédé réel. L'étude du système informatique distribué esquissé dans ce mémoire nous semble également un thème de recherche à approfondir.

Le calculateur vecteur a été conçu spécialement pour le calcul de la commande optimale de processus mais l'architecture bien que simple offre des possibilités intéressantes. Il nous est apparu qu'il pouvait permettre d'envisager la résolution d'une vaste gamme de problèmes demandant des calculs vectoriels.

BIBLIOGRAPHIE

- 1 F. LHOTE, J.C. MIELLOU, P. COMPTE, J.M. HENRIOUD, B. LANG, p. SPITERI
"Algorithmes de relaxation décentralisation en contrôle optimal"
ATP "Commande" CNRS, 75
- 2 F. LHOTE, J.C. MIELLOU, B. LANG, P. SPITERI
"Mise au point d'un système hybride de commande optimale décentralisée"
ATP "Systèmes automatiques complexes" CNRS, 78
- 3 F. LHOTE, J.C. MIELLOU
"Algorithmes de décentralisation et de coordination par relaxation en commande optimale"
Analyse et commande de systèmes complexes, Monographie AFCET,
Editions Cepadues - Toulouse, pp.133-143, 79
- 4 F. LHOTE, B. LANG, J.C. MIELLOU, P. SPITERI
"Decomposition and coordination by relaxation methods for parallel, in line, computation of the optimal control large systems"
9th IFIP Conference on Optimisation Techniques, Varsovie, 79
- 5 B. LANG
"Calcul hybride décentralisé de la commande optimale d'un processus dynamique continu"
Thèse de Docteur Ingénieur, Besançon 77
- 6 N.E. CHEICKH OBEID
"Commande optimale en boucle fermée par intermittence de systèmes multivariables, application à un processus thermique"
Thèse de Troisième Cycle, Besançon 80
- 7 A. AUSLENDER
"Optimisation, Méthodes numériques"
Masson, 76

- 8 L. PUN
"Introduction à la pratique de l'optimisation"
Dunod, 72
- 9 R. BOUDAREL, J. DELMAS, P. GUICHET
"Commande optimale des processus"
Tome 2, Dunod, 68
- 10 M.G. SINGH, A. TITLI
"Systems ; Décomposition, Optimisation and control"
Pergamon Press, 78
- 11 M.D. MESAROVIC, D. MACKO, Y. TAKAHARA
"Theory of hierarchical multilevel systems"
Academic Press, 70
- 12 A. TITLI
"Commande hiérarchisée et optimisation des processus complexes"
Dunod, 75
- 13 P. BERNHARD
"Commande optimale, décentralisation et jeux dynamiques"
Dunod, 76
- 14 E. DURAND
"Solutions numériques des équations algébriques"
Tome 2, Masson, 72
- 15 W.L. MIRANKER
"Parallel methods for solving equations"
Mathematic and computers in simulation - Vol.XX, number 2,
June 78, pp.93-101, North Holland
- 16 J.C. MIELLOU
"Algorithmes de relaxation chaotique à retards"
RAIRO, série Rouge, R1 (75), pp.55-82

- 17 J.A. JENSEN, J.H. ROWLAND
"Methods of computation"
Scott, Foresman and Company, 75
- 18 N. GASTINEL
"Analyse numérique linéaire"
Hermann, 66
- 19 K.J. THURBER
"Parallel Processor architectures"
Computer Design, January 79, pp.89-97
February 79, pp.103-114
- 20 M.J. FLYNN
"Some Computer Organisations and their effectiveness"
I.E.E.E. Transactions on computers, September 72, pp.948-960
- 21 A. GANDOSI
"Le calculateur parallèle et associatif PROPAL 2"
1 st European Conference on Parallel and distributed Processing,
Editions Cepadues, Toulouse 79
- 22 G.H. BARNIER
"The Illiac IV Computer"
I.E.E.E. Transactions on Computers, August 68, pp.97-107
- 23 J. STRELCHUN
"Array Processor responds in Real Time"
Electronics, August 79
- 24 H. ARBEY, F. GIRER, J.P. SCHOEFFER
"Micro-calculateur - Multi-processeur"
Travail de projet, ENSMM, 77
- 25 G. BOULAYE
La microprogrammation
Dunod, 71

